DB2

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Data Recovery and High Availability Guide and Reference

SC10-4228-00

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Data Recovery and High Availability Guide and Reference

SC10-4228-00

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

Contents

About this book

This book provides detailed information about, and shows you how to use, the IBM DB2 database 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 database 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 the DB2 database system, Structured Query Language (SQL), and with the operating system environment in which the DB2 database system 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 2, "Database backup"
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 recreate 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.

Chapter 5, "Database recover"
Describes the DB2 recover utility, which performs the neccessary restore and rollforward operations to recover a database to a specified time, based on information found in the recovery history file.

High Availability

Chapter 6, "Introducing high availability and failover support"
Presents an overview of the high availability failover support that is provided by DB2.

Chapter 7, "High availability disaster recovery (HADR)"
Discusses the concepts and procedures required to set up and manage a high availability disaster recovery (HADR) environment.

Chapter 8, "Cluster support 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 9, "Cluster support 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 10, "Cluster support for 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 programs"
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 G, "Tivoli Storage Manager"
Provides information about the Tivoli Storage Manager (TSM) product, which you can use to manage database or table space backup operations.

Appendix H, "Tivoli Space Manager Hierarchical Storage Management support for partitioned tables"
Provides information about using the Tivoli® Space Manager Hierarchical Storage Manager (HSM) client program automatically migrates eligible files to secondary storage to maintain specific levels of free space on local file systems.

Appendix F, "Cross-node recovery with the db2adutl command and the logarchopt1 and vendoropt database configuration parameters"
Provides examples that show how to perform cross-node recovery using the db2adutl command, and the *logarchopt1* and *vendoropt* database configuration parameters.

Appendix I, "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 J, "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 might, at one time or another, encounter storage problems, power interruptions, or application failures, and each failure scenario requires a different recovery action. 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?
- Should I configure a standby system, either manually or through high availability disaster recovery (HADR)?

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 environments, 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 recreate the database if it becomes damaged or corrupted in some way.

The recreation 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.

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.

You cannot directly modify the recovery history file or the table space change history file; however, you can delete entries from the files using the PRUNE HISTORY command. You can also use the *rec_his_retentn* database configuration parameter to specify the number of days that these history files will be retained.

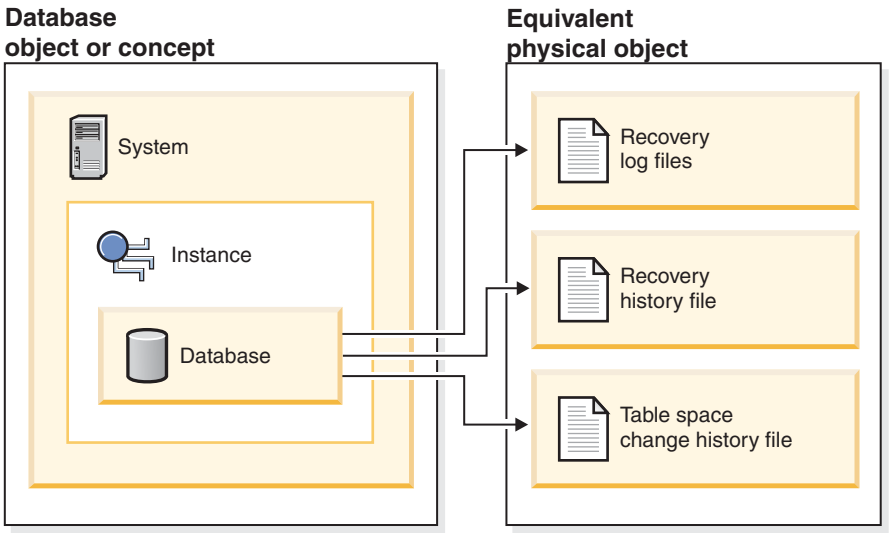


Figure 1. Database recovery files

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. If both the *logarchmeth1* and *logarchmeth2* database configuration parameters are set to "OFF" then the database is *Non-recoverable*. 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 the *logarchmeth1* or *logarchmeth2* database configuration parameters set to a value other than "OFF". 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). Online table space restore and rollforward operations are supported only if the database is recoverable. If the database is non-recoverable, database restore and rollforward operations must 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.

Automated backup operations:

Since it can be time-consuming to determine whether and when to run maintenance activities such as backup operations, you can use the Configure Automatic Maintenance wizard to do this for you. With automatic maintenance, you specify your maintenance objectives, including when automatic maintenance can run. DB2® then uses these objectives to determine if the maintenance activities need to be done and then runs only the required maintenance activities during the next available maintenance window (a user-defined time period for the running of automatic maintenance activities).

Note: You can still perform manual backup operations when automatic maintenance is configured. DB2 will only perform automatic backup operations if they are required.

Related concepts:

- "Crash recovery" on page 10
- "High availability disaster recovery overview" on page 221
- "Rollforward recovery" on page 24
- "Version recovery" on page 23

Related reference:

- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*
- "rec_his_retentn - Recovery history retention period configuration parameter" in *Performance Guide*

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 might 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). To recover a database, you can use either a full database backup image that contains all of the table space backup images, or you can rebuild the database using selected table space images. Table space backup images are also useful for recovering from an isolated disk failure or an application error. In partitioned database environments, you only need to restore the table spaces that reside on database partitions that have failed. You do not need to restore all of the table spaces or all of the database partitions.

Although full database backups are no longer required for database recovery now that you can rebuild a database from table space images, it is still good practice to occasionally take a full backup of your database.

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 restore 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 might 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:

- "Developing a backup and recovery strategy" on page 3
- "Incremental backup and recovery" on page 27

Related reference:

- "Configuration parameters for database logging" on page 37
- Appendix I, "User exit for database recovery," on page 409

Storage considerations for recovery

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 might decide to use a recovery plan that only occasionally saves a backup of this table space. You can 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 restore 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:

- Specify a log archiving method using the LOGARCHMETH1 or LOGARCHMETH2 configuration parameters.
- 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.
- Use a user exit program to copy these logs to another storage device in your environment.

Related concepts:

- "Log file management through log archiving" on page 49

Related reference:

- "Configuration parameters for database logging" on page 37
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*

- "logarchmeth2 - Secondary log archive method configuration parameter" in *Performance Guide*

Keeping related data together

In the process of designing your database, you will develop an understanding of the relationships that exist between tables. These relationships can be expressed:

- At the application level, when transactions update more than one table
- 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.

Related concepts:

- "Crash recovery" on page 10
- "Developing a backup and recovery strategy" on page 3
- "Disaster recovery" on page 22
- "Version recovery" on page 23

Related tasks:

- "Using rollforward" on page 157

Backup and restore operations between different operating systems and hardware platforms

DB2 database systems support some backup and restore operations between different operating systems and hardware platforms.

The supported platforms for DB2 backup and restore operations can be grouped into one of three families:

- Big-endian Linux® and UNIX®
- Little-endian Linux and UNIX
- Windows

A database backup from one platform family can be restored on any system within the same platform family. For Windows operating systems, you can restore a database created on DB2 UDB V8 on a DB2 Version 9 database system. For Linux and UNIX operating systems, as long as the endianness (big endian or little endian) of the backup and restore platforms is the same, you can restore backups that were produced on DB2 UDB V8 on DB2 Version 9.

The following table shows each of the Linux and UNIX platforms DB2 supports and indicates whether the platforms are big endian or little endian:

Table 1. Endianness of supported Linux and UNIX operating systems DB2 supports

Platform	Endianness
AIX®	big endian

Table 1. Endianness of supported Linux and UNIX operating systems DB2 supports (continued)

Platform	Endianness
HP-UX	big endian
HP on IPF	big endian
Solaris Operating environment	big endian
Linux on zSeries	big endian
Linux on Power PC	big endian
Linux on IA-64	little endian
Linux on AMD64 and Intel® EM64T	little endian
32-bit Linux on x86	little endian

The target system must have the same (or later) version of the DB2 database product as the source system. You cannot restore a backup created on one version of the database product to a system running an earlier version of the database product. For example, you can restore a V8 backup on a V9 database system, but you cannot restore a V9 backup on a V8 database system.

In situations where certain backup and restore combinations are not allowed, you can move tables between DB2 databases using other methods:

- db2move command
- Export utility followed by the import or the load utilities

Related tasks:

- "Using backup" on page 66
- "Using restore" on page 90

Related reference:

- "db2move - Database movement tool command" in *Command Reference*
- "EXPORT command" in *Command Reference*
- "IMPORT Command" in *Command Reference*
- "LOAD command" in *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 on page 11). When a database is in a consistent and usable state, it has attained what is known as a "point of consistency".

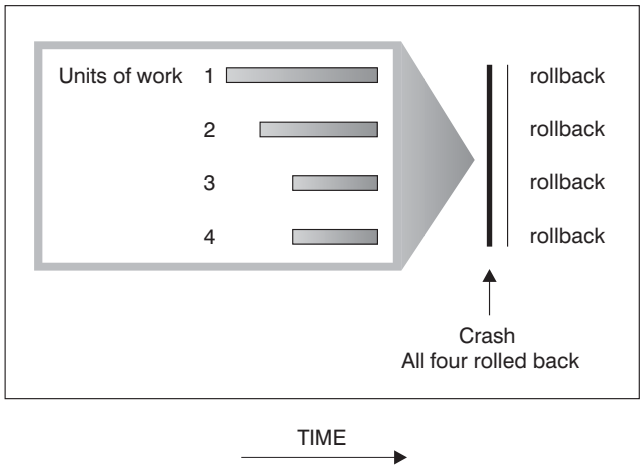


Figure 2. Rolling Back Units of Work (Crash Recovery)

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 hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes DB2 to go down
- An application terminating abnormally.

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 *logarchmeth1* configuration parameter is not set to `OFF`), 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:

- "autorestart - Auto restart enable configuration parameter" in *Performance Guide*
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*

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.

  Notes:

  1. Table space restore is only valid for recoverable databases, because the database must be rolled forward.
  2. If you restore the catalog table space, you must perform a rollforward operation to the end of logs.

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 want, 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 might have to set the *indexrec* configuration parameter to `ACCESS` to avoid restart failures.

Related tasks:

- "Recovering table spaces in non-recoverable databases" on page 13
- "Recovering table spaces in recoverable databases" on page 12

Related reference:

- "RESTART DATABASE command" in *Command Reference*
- "RESTORE DATABASE " on page 100

Recovering table spaces in recoverable databases

When crash recovery is necessary, the damaged table spaces is taken offline and is not accessible. It is placed in roll-forward pending state. If there are no additional problems, a restart operation will succeed in bringing the database online even with the damaged table space. Once online, the damaged table space is unusable, but the rest of the database is usable. To fix the damaged table space and make it useable, follow the procedure below.

Procedure:

To make the damaged table space useable, use one of the procedures that follow:

- Method 1
  1. Fix the damaged containers without losing the original data.
  2. Complete a table space rollforward operation to the end of the logs.

     Note: The rollforward operation will first attempt to bring the table space from offline to normal state.
- Method 2
  1. Fix the damaged containers with or without losing the original data.
  2. Perform a table space restore operation.
  3. Complete a table space rollforward operation to the end of the logs or to a point-in-time.

Related concepts:

- "Recovering damaged table spaces" on page 12
- "Understanding table space states" on page 60

Related tasks:

- "Recovering table spaces in non-recoverable databases" on page 13

Related reference:

- "RESTART DATABASE command" in *Command Reference*

Recovering table spaces in non-recoverable databases

When crash recovery is needed, but there are damaged table spaces, you can only successfully restart the database if the damaged table spaces are dropped. In a non-recoverable database, the logs necessary to recover the damaged table spaces are not retained. Therefore, the only valid action against such table spaces is to drop them.

Procedure:

To restart a database with damaged table spaces:

1. Invoke an unqualified restart database operation. It will succeed if there are no damaged table spaces. If it fails (SQL0290N), look in the administration notification log file for a complete list of table spaces that are currently damaged.
2. If you are willing to drop all of the damaged table spaces, 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 you must drop the table space after the recovery operation is complete.

   The restart operation continues without recovering the specified table spaces. If a damaged table space is *not* included in the DROP PENDING TABLESPACES list, the restart database operation fails with SQL0290N.

Note: Including a table space name in the DROP PENDING TABLESPACES list does not mean that the table space will be in drop pending state. It will be placed in this state only if the table space is found to be damaged during the restart operation.

3. If the restart database operation is successful, invoke the LIST TABLESPACES command to find out which table spaces are in drop pending state.
4. Issue DROP TABLESPACE statements to drop each of the table spaces that are in drop pending state. Once you have done this, you will be able to reclaim the space that the damaged table spaces were using or recreate the table spaces.
5. If you are unwilling to drop and lose the data in the damaged table spaces, you can:
   - Fix the damaged containers (without losing the original data).
   - Reissue the RESTART DATABASE command.
   - Perform a database restore operation.

Related concepts:

- "Recovering damaged table spaces" on page 12
- "Understanding table space states" on page 60

Related tasks:

- "Recovering table spaces in recoverable databases" on page 12

Related reference:

- "RESTART DATABASE command" in *Command Reference*

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 partition. Because this database partition 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. To do this, use the *mirrorlogpath* configuration parameter to specify a mirror log path on a RAID level 1 file system. 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.

Related concepts:

- "Backup overview" on page 63
- "Developing a backup and recovery strategy" on page 3

Reducing the impact of transaction failure

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

- An uninterrupted power supply for each DB2 server
- Adequate disk space for database logs on all database partitions
- 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 environment" on page 166

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 partition, 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 partition 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 partition begins the second phase.

During the second phase, the coordinator partition 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 partition. 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 partition 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 partition 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 partition 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 partition, agents working for this application are interrupted. The coordinator partition 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 partition 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 database partition.

If the database manager ends abnormally, database partitions on the server can 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 partition, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator partition, 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 partition for an application:

- If the server that restarted is not the coordinator partition 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 partition 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 might not be able to resolve all the indoubt transactions (for example, some of the database partition servers might 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 Enterprise Server 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 partition for the transaction.

SQL1229N
This SQLCODE is received when the database partition server that failed is not the coordinator partition 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 can cause other logical nodes on the same processor to fail.

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

Related tasks:
- "Resolving indoubt transactions manually" in *Administration Guide: Planning*

Related reference:
- "db2start - Start DB2 command" in *Command Reference*
- "LIST INDOUBT TRANSACTIONS command" in *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 command" in *Command Reference*
- "RESTART DATABASE command" in *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 iSeries™ database server during a transaction, there are some differences in how indoubt transactions are recovered.

To access host or iSeries 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 iSeries servers is normally performed automatically by the Transaction Manager (TM) and the DB2 Syncpoint Manager (SPM). An indoubt transaction at a host or iSeries server does not hold any resources at the local DB2 location, but does hold resources at the host or iSeries server as long as the transaction is indoubt at that location. If the administrator of the host or iSeries server determines that a heuristic decision must be made, then the administrator might contact the local DB2 database administrator (for example via telephone) to determine whether to commit or roll back the transaction at the host or iSeries 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 iSeries server. The partner_lu is the fully qualified luname of the host or iSeries server. This provides the best identification of the host or iSeries server, and should be provided by the caller from the host or iSeries server. The luwid provides a unique identifier for a transaction and is available at all hosts and iSeries 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 can 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 iSeries 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 21

Related reference:

- "db2start - Start DB2 command" in *Command Reference*
- "LIST INDOUBT TRANSACTIONS command" in *Command Reference*
- "RESTART DATABASE command" in *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 iSeries database server during a transaction, there are some differences in how indoubt transactions are recovered.

To access host or iSeries database servers, DB2 Connect is used. The recovery steps differ if DB2 Connecthas the DB2 Syncpoint Manager configured.™

Procedure:

Use the information in this section when TCP/IP connectivity is used to update DB2 for z/OS® 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 z/OS 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 z/OS 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 THREAD command, see the *DB2 for z/OS Command Reference*.

Related tasks:

- "Recovering indoubt transactions on the host when DB2 Connect has the DB2 Syncpoint Manager configured" on page 20

Related reference:

- "LIST INDOUBT TRANSACTIONS command" in *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 either database backups, table space backups, or both, as well as archived logs.

If your plan for disaster recovery is to restore the entire database on another machine, it is recommended that you have at least one full database backup and all the archived logs for the database. Although it is possible to rebuild a database if you have a full table space backup of each table space in the database, this method might involve numerous backup images and be more time-consuming than recovery using a full database backup.

You can choose to keep a standby database up to date by applying the logs to it as they are archived. Or, you can choose to keep the database or table space backups and log archives in the standby site, and perform restore and rollforward operations only after a disaster has occurred. (In the latter case, recent backup images are preferable.) In a disaster situation, 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 is less complicated and time-consuming if you restore the entire database; therefore, a full database backup should be kept at a standby site. 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.

Another way you can protect your data from partial or complete site failures is to implement the DB2 high availability disaster recovery (HADR) feature. Once it is set up, HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby.

You can also protect your data from partial or complete site failures using replication. Replication allows you to copy data on a regular basis to multiple remote databases. DB2 database provides a number of replication tools that allow you to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied.

Storage mirroring, such as Peer-to-Peer Remote Copy (PPRC), can also be used to protect your data. PPRC provides a synchronous copy of a volume or disk to protect against disasters.

DB2 provides you with several options when planning for disaster recovery. Based on your business needs, you might decide to use table space or full database backups as a safeguard against data loss, or you might decide that your environment is better suited to a solution like HADR. Whatever your choice, you should test your recovery procedures in a test environment before implementing them in your production environment.

Related concepts:

- "High availability disaster recovery overview" on page 221
- "Redefining table space containers during a restore operation (redirected restore)" on page 94
- "Using replication to move data" in *Data Movement Utilities Guide and Reference*

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 restore 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).

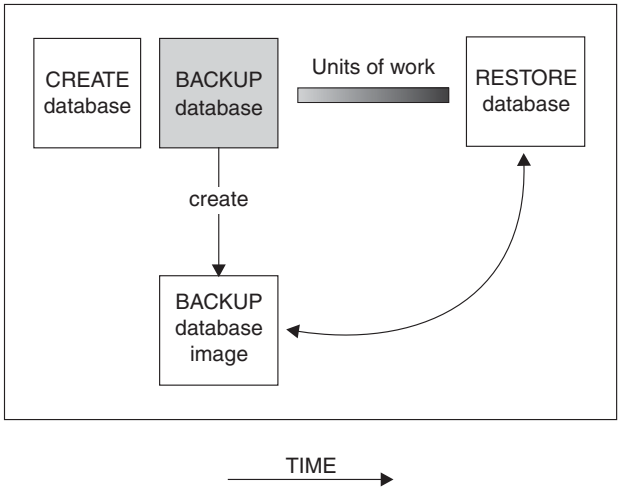


Figure 3. Version Recovery. Shows that units of work from the time of the backup to the time of the failure is lost.

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 database 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*.

Related concepts:

- "Crash recovery" on page 10
- "Disaster recovery" on page 22

Rollforward recovery

To use the *rollforward recovery* method, you must have taken a backup of the database and archived the logs (by setting the *logarchmeth1* and *logarchmeth2* configuration parameters to a value other than OFF). 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:

- You are restoring from an online backup image
- You are issuing a table space-level restore

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 on page 25). 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, and the ROLLFORWARD DATABASE command must be issued on the database partition where the catalog tables for the database resides (catalog partition). If you are performing point-in-time rollforward recovery, all database partitions must be rolled forward to ensure that all database 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 database 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.

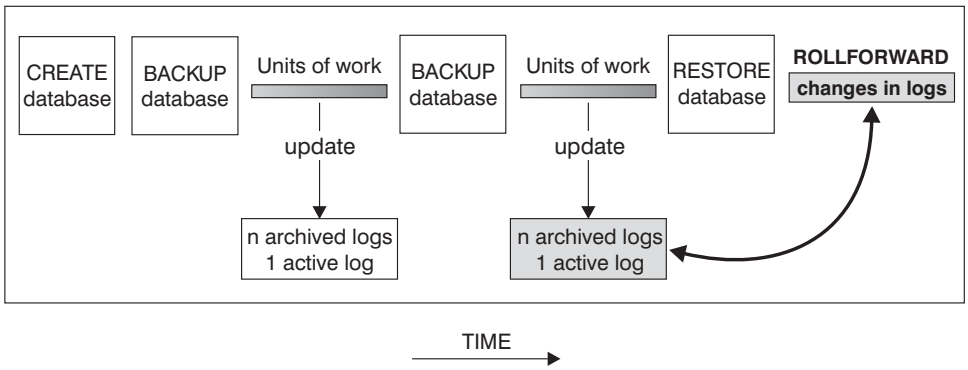


Figure 4. Database Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

- *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 26). 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 might 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.

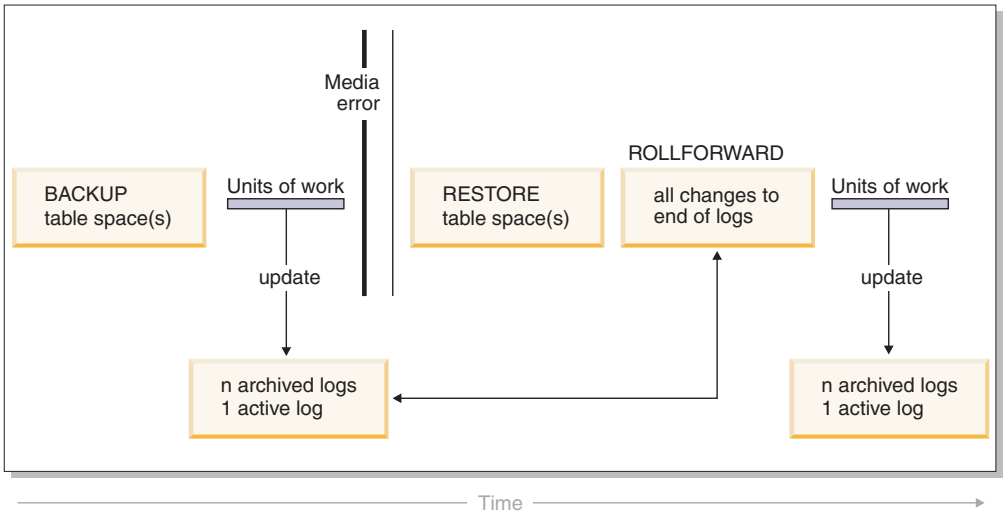


Figure 5. Table Space Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

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 database partitions on which the table space resides. DB2 submits the rollforward request to all database 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 database partitions. If you want to roll all table spaces (on all database 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 database partitions.

If you are rolling a table space forward that contains any piece of a partitioned table and you are rolling it forward to a point in time, you must also roll all of the other table spaces in which that table resides forward to the same point in time. However, you can roll a single table space containing a piece of a partitioned table forward to the end of logs.

Note: If a partitioned table has any attached, detached, or dropped data partitions, then point-in-time rollforward must also include all table spaces for these data partitions. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSDATAPARTITIONS catalog table.

Related concepts:

- "Understanding recovery logs" on page 33

Related reference:

- "ROLLFORWARD DATABASE " on page 168

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.

To address these issues, DB2 provides incremental backup and recovery. 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.

Notes:

1. If a table space contains long field or large object data and an incremental backup is taken, all of the long field or large object data will be copied into the backup image if any of the pages in that table space have been modified since the previous backup.
2. If you take an incremental backup of a table space that contains a dirty page (that is, a page that contains data that has been changed but has not yet been written to disk) then all large object data is backed up. Normal data is backed up only if it has changed.

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 restore 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 28

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 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.
- 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).

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 20031228152133
```

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 might 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.

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 performing 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.

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.

Related concepts:

- "Incremental backup and recovery" on page 27

Related reference:

- "db2ckrst - Check incremental restore image sequence " on page 314
- "RESTORE DATABASE " on page 100

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 space 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 27

Related tasks:

- “Restoring from incremental backup images” on page 28

Related reference:

- “RESTORE DATABASE ” on page 100

Monitoring the progress of backup, restore and recovery operations

You can use the LIST UTILITIES command to monitor backup, restore, and rollforward operations on a database.

Procedure:

To use progress monitoring for backup, restore and recovery operations:

- Issue the LIST UTILITIES command and specify the SHOW DETAIL option.

```
list utilities show detail
```

The following is an example of the output for monitoring the performance of an offline database backup operation:

```
LIST UTILITIES SHOW DETAIL

ID                        = 2
Type                      = BACKUP
Database Name             = SAMPLE
Description               = offline db
Start Time                = 10/30/2003 12:55:31.786115
Throttling:
Priority                  = Unthrottled
Progress Monitoring:
Estimated Percentage Complete    = 41
      Total Work Units     = 20232453 bytes
      Completed Work Units = 230637 bytes
      Start Time           = 10/30/2003 12:55:31.786115
```

For backup operations, an initial estimate of the number of bytes to be processed will be specified. As the backup operation progresses the number of bytes to be processed will be updated. The bytes shown does not correspond to the size of the image and should not be used as an estimate for backup image size. The actual image might be much smaller depending on whether it is an incremental or compressed backup.

For restore operations, no initial estimate will be given. Instead UNKNOWN will be specified. As each buffer is read from the image, the actual amount of bytes read will be updated. For automatic incremental restore operations where multiple images might be restored, the progress will be tracked using phases. Each phase represents an image to be restored from the incremental chain. Initially, only one phase will be indicated. After the first image is restored, the total number of phases will be indicated. As each image is restored the number of phases completed will be updated, as will the number of bytes processed.

For crash recovery and rollforward recovery there will be two phases of progress monitoring: FORWARD and BACKWARD. During the FORWARD phase, log files are read and the log records are applied to the database. For crash recovery, the total amount of work is estimated using the starting log sequence number up to the end of the last log file. For rollforward recovery, when this phase begins UNKNOWN will be specified for the total work estimate. The amount of work processed in bytes will be updated as the process continues.

During the BACKWARD phase, any uncommitted changes applied during the FORWARD phase are rolled back. An estimate for the amount of log data to be processed in bytes will be provided. The amount of work processed in bytes will be updated as the process continues.

Related concepts:

- "Backup overview" on page 63
- "Crash recovery" on page 10
- "Restore overview" on page 89
- "Rollforward overview" on page 155

Related reference:

- "LIST UTILITIES command" in *Command Reference*

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 *logarchmeth1* and *logarchmeth2* database configuration parameters are set to `OFF`.) 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.

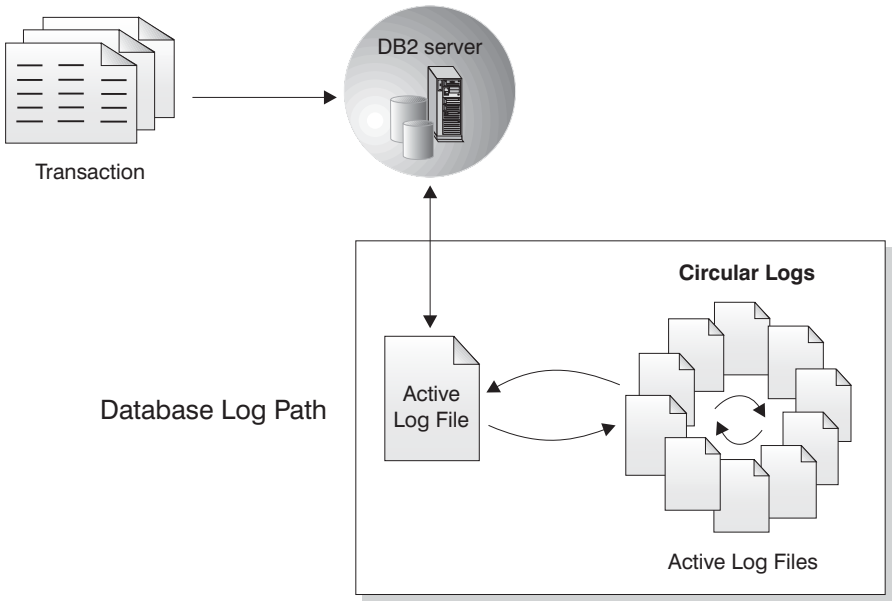


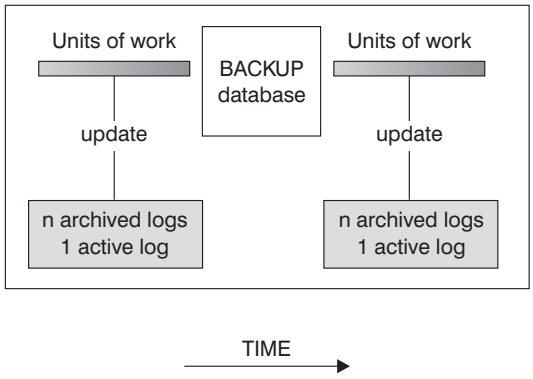
Figure 6. Circular Logging

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. Active logs are located in the database log path directory.

- *Archive* logging is used specifically for rollforward recovery. Archived logs are logs that were active but are no longer required for crash recovery. Use the *logarchmeth1* database configuration parameter to enable archive logging.

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to restore 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.



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.

The following database configuration parameters allow you to change where archived logs are stored: The *newlogpath* parameter, and the *logarchmeth1* and *logarchmeth2* parameters. 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 *Performance Guide* book.

Related concepts:

- "Log mirroring" on page 35

Related reference:

- "Configuration parameters for database logging" on page 37
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*
- "loghead - First active log file configuration parameter" in *Performance Guide*
- Appendix I, "User exit for database recovery," on page 409

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 might be damaged (as a result of a disk crash), you should consider using the MIRRORLOGPATH configuration parameter 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 either full or truncated. 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 concepts:

- "Log file management through log archiving" on page 49
- "Standby database states in high availability disaster recovery (HADR)" on page 226

Related reference:

- "mirrorlogpath - Mirror log path configuration parameter" in *Performance Guide*
- "hadr_primary_log_file - HADR Primary Log File monitor element" in *System Monitor Guide and Reference*
- "hadr_standby_log_file - HADR Standby Log File monitor element" in *System Monitor Guide and Reference*

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 might 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 can 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 might 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).
- If you are using high availability disaster recovery (HADR) you should not use the NOT LOGGED INITIALLY table attribute. Tables created with the NOT LOGGED INITIALLY option specified are not replicated. Attempts access such tables after an HADR standby database takes over as the primary database will result in an error.
- 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 *SQL Reference, Volume 1*

Related tasks:

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

Related reference:

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

Configuration parameters for database logging

Archive Retry Delay (archretrydelay)
Specifies the amount of time (in seconds) to wait between attempts to archive log files after the previous attempt fails. The default value is 20.

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 might 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.

Failover Archive Path (failarchpath)
Specifies an alternate directory for the archive log files if the log archive method specified fails. This directory is a temporary storage area for the log files until the log archive method that failed becomes available again at

which time the log files will be moved from this directory to the log archive method. By moving the log files to this temporary location, log directory full situations might be avoided. This parameter must be a fully qualified existing directory.

Log archive method 1 (logarchmeth1), log archive method 2 (logarchmeth2)
These parameters cause the database manager to archive log files to a location that is not the active log path. If both of these parameters are specified, each log file is archived twice. This means that you will have two copies of archived log files in two different locations.

Valid values for these parameters include a media type and, in some cases, a target field. Use a colon (:) to separate the values. Valid values are:

OFF Specifies that the log archiving method is not to be used. If both *logarchmeth1* and *logarchmeth2* are set to `OFF`, the database is considered to be using circular logging and will not be rollforward recoverable. This is the default.

LOGRETAIN
This value can only be used for *logarchmeth1* and is equivalent to setting the *logretain* configuration parameter to `RECOVERY`. If you specify this value, the *logretain* configuration parameters will automatically be updated.

USEREXIT
This value is only valid for *logarchmeth1* and is equivalent to setting the *userexit* configuration parameter to `ON`. If specify this value, the *userexit* configuration parameter will be automatically updated.

DISK This value must be followed by a colon(:) and then a fully qualified existing path name where the log files will be archived. For example, if you set *logarchmeth1* to `DISK:/u/dbuser/archived_logs` the archive log files will be placed in a directory called `/u/dbuser/archived_logs`.

Note: If you are archiving to tape, you can use the db2tapemgr utility to store and retrieve log files.

TSM If specified without any additional configuration parameters, this value indicates that log files should be archived on the local TSM server using the default management class. If followed by a colon(:) and a TSM management class, the log files will be archived using the specified management class.

VENDOR
Specifies that a vendor library will be used to archive the log files. This value must be followed by a colon(:) and the name of the library. The APIs provided in the library must use the backup and restore APIs for vendor products.

Notes:

1. If either *logarchmeth1* or *logarchmeth2* is set to a value other than `OFF`, the database is configured for rollforward recovery.
2. If you update the *userexit* or *logretain* configuration parameters *logarchmeth1* will automatically be updated and vice versa. However, if you are using either *userexit* or *logretain*, *logarchmeth2* must be set to `OFF`.

Log archive options 1 (logarchopt1), log archive options 2 (logarchopt2)
Specifies a string which is passed on to the TSM server or vendor APIs. For TSM, this field is used to allow the database to retrieve logs that were generated on a different TSM node or by a different TSM user. The string must be provided in the following format:

```
"-fromnode=nodename -fromowner=ownername"
```

where `nodename` is the name of the TSM node that originally archived the log files, and `ownername` is the name of the TSM user that originally archived the log files. Each log archive options field corresponds to one of the log archive methods: *logarchopt1* is used with *logarchmeth1*, and *logarchopt2* is used with *logarchmeth2*.

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. However, recovery can take longer with a larger log buffer size value.

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 can increase the chance of failure or cause a delay in log shipping scenarios. When considering active log space, it might 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 retain (logretain)
This configuration parameter has been replaced by *logarchmeth1*. It is still supported for compatibility with previous versions of DB2.

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.

Note: The default value for the *logretain* database configuration parameter does not support rollforward recovery. You must change the value of this parameter if you are going to use rollforward recovery.

Maximum log per transaction (max_log)
This parameter indicates the percentage of primary log space that can be consumed by one transaction. The value is a percentage of the value specified for the logprimary configuration parameter.

If the value is set to `0`, there is no limit to the percentage of total primary log space that a transaction can consume. If an application violates the *max_log* configuration, the application will be forced to disconnect from the database, the transaction will be rolled back, and error SQL1224N will be returned.

You can override this behavior by setting the DB2_FORCE_APP_ON_MAX_LOG registry variable to *FALSE*. This will cause transactions that violate the *max_log* configuration to fail and return error SQL0964N. The application can still commit the work completed by previous statements in the unit or work, or it can roll the work completed back to undo the unit of work.

This parameter, along with the num_log_span configuration parameter, can be useful when infinite active logspace is enabled. If infinite logging is on (that is, if logsecondary is -1) then transactions are not restricted to the upper limit of the number of log files (logprimary + logsecond). When the value of logprimary is reached, DB2 starts to archive the active logs, rather than failing the transaction. This can cause problems if, for instance, there is a long running transactions that has been left uncommitted (perhaps caused by a bad application). If this occurs, the active logspace keeps growing, which might lead to poor crash recovery performance. To prevent this, you can specify values for either one or both of the max_log or num_log_span configuration parameters.

Note: The following DB2 commands are excluded from the limitation imposed by the *max_log* configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG TABLE (online), RESTORE DATABASE, and ROLLFORWARD 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 might 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.

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 might 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.

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.

Number of archive retries on error (numarchretry)
Specifies the number of attempts that will be made to archive log files using the specified log archive method before they are archived to the path

specified by the *failarchpath* configuration parameter. This parameter can only be used if the *failarchpath* configuration parameter is set. The default value is 5.

Number log span (num_log_span)
This parameter indicates the number of active log files that an active transaction can span. If the value is set to 0, there is no limit to how many log files one single transaction can span.

If an application violates the *num_log_span* configuration, the application will be forced to disconnect from the database and error SQL1224N will be returned.

This parameter, along with the max_log configuration parameter, can be useful when infinite active logspace is enabled. If infinite logging is on (that is, if logsecondary is -1) then transactions are not restricted to the upper limit of the number of log files (logprimary + logsecond). When the value of logprimary is reached, DB2 starts to archive the active logs, rather than failing the transaction. This can cause problems if, for instance, there is a long running transactions that has been left uncommitted (perhaps caused by a bad application). If this occurs, the active logspace keeps growing, which might lead to poor crash recovery performance. To prevent this, you can specify values for either one or both of the max_log or num_log_span configuration parameters.

Note: The following DB2 commands are excluded from the limitation imposed by the *num_log_span* configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG TABLE (online), RESTORE DATABASE, and ROLLFORWARD DATABASE.

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 database partition number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

Primary logs (logprimary)

This parameter specifies the number of primary logs of size *logfilsiz* 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. Infinite active logging is useful in environments that must accommodate large jobs requiring more log space than you would normally allocate to the primary logs.

Notes:

1. Log archiving must be enabled in order to set *logsecond* to -1.
2. If this parameter is set to -1, crash recovery time might be increased since DB2 might need to retrieve archived log files.

User exit (userexit)

This configuration parameter has been replaced by *logarchmeth1*. It is still supported for compatibility with previous versions of DB2.

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

might 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 might have design limits that restrict the number of operations per day. These factors must be considered when you determine the log size.

Notes:

1. This value must be set to `ON` to enable infinite active log space.
2. The default value for the *userexit* database configuration parameter does not support rollforward recovery, and must be changed if you are going to use it.

Related concepts:

- "Enhancing recovery performance" on page 61
- "Log archiving configuration for high availability disaster recovery (HADR)" on page 257
- "Log file management" on page 46

Related reference:

- "blk_log_dsk_ful - Block on log disk full configuration parameter" in *Performance Guide*
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*
- "logarchmeth2 - Secondary log archive method configuration parameter" in *Performance Guide*
- "logbufsz - Log buffer size configuration parameter" in *Performance Guide*
- "logfilsiz - Size of log files configuration parameter" in *Performance Guide*
- "logprimary - Number of primary log files configuration parameter" in *Performance Guide*
- "logretain - Log retain enable configuration parameter" in *Performance Guide*
- "logsecond - Number of secondary log files configuration parameter" in *Performance Guide*
- "max_log - Maximum log per transaction configuration parameter" in *Performance Guide*
- "mincommit - Number of commits to group configuration parameter" in *Performance Guide*
- "mirrorlogpath - Mirror log path configuration parameter" in *Performance Guide*
- "newlogpath - Change the database log path configuration parameter" in *Performance Guide*
- "num_log_span - Number log span configuration parameter" in *Performance Guide*

- "overflowlogpath - Overflow log path configuration parameter" in *Performance Guide*
- "userexit - User exit enable configuration parameter" in *Performance Guide*
- Appendix I, "User exit for database recovery," on page 409
- "db2tapemgr - Manage log files on tape command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Configuring database logging options

Use database logging configuration parameters to specify data logging options for your database, such as the type of logging to use, the size of the log files, and the location where log files should be stored.

Prerequisites:

To configure database logging options, you must have SYSADM, SYSCTRL, or SYSMAINT authority.

Procedure:

You can configure database logging options using the UPDATE DATABASE CONFIGURATION command on the command line processor (CLP), through the Configure Database Logging wizard GUI in the Control Center, or by calling the db2CfgSet API.

To configure database logging options using the UPDATE DATABASE CONFIGURATION command on the command line processor:

1. Specify whether you want to use circular logging or archive logging. If you want to use circular logging, the LOGARCHMETH1 and LOGARCHMETH2 database configuration parameters must be set to OFF. This is the default setting. To use archive logging, you must set at least one of these database configuration parameters to a value other than OFF. For example, if you want to use archive logging and you want to save the archived logs to disk, issue the following:

   ```
   db2 update db configuration for mydb using logarchmeth1
      disk:/u/dbuser/archived_logs
   ```

   The archived logs will be placed in a directory called /u/dbuser/archived_logs.
2. Specify values for other database logging configuration parameters, as required. The following are additional configuration parameters for database logging:
   - ARCHRETRYDELAY
   - BLK_LOG_DSK_FUL
   - FAILARCHPATH
   - LOGARCHOPT1
   - LOGARCHOPT2
   - LOGBUFSZ
   - LOGFILSIZ
   - LOGPRIMARY
   - LOGRETAIN
   - LOGSECOND

- MAX_LOG
- MIRRORLOGPATH
- NEWLOGPATH
- MINCOMMIT
- NUMARCHRETRY
- NUM_LOG_SPAN
- OVERFLOWLOGPATH
- USEREXIT

For more information on these database logging configuration parameters, refer to Configuration parameters for database logging.

To open the Configure Database Logging wizard:

1. From the Control Center, expand the object tree until you find the database for which you want to set up logging.
2. Right-click on the database and select Configure Database Logging from the pop-up menu. The Configure Database Logging wizard opens.

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

Related reference:

- "Configuration parameters for database logging" on page 37

Log file management

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 file names after restoring a database (with or without rollforward recovery). The database manager ensures that an incorrect log is not applied during rollforward recovery. If DB2 reuses a log file name after a restore operation, the new log files are archived to separate directories so that multiple log files with the same name can be archived. The location of the log files is recorded in the recovery history file so that they can be applied during rollforward recovery. 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 might 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 can be associated with two or more different *log sequences* for a database, because log file names are reused (see Figure 8). 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 might 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.

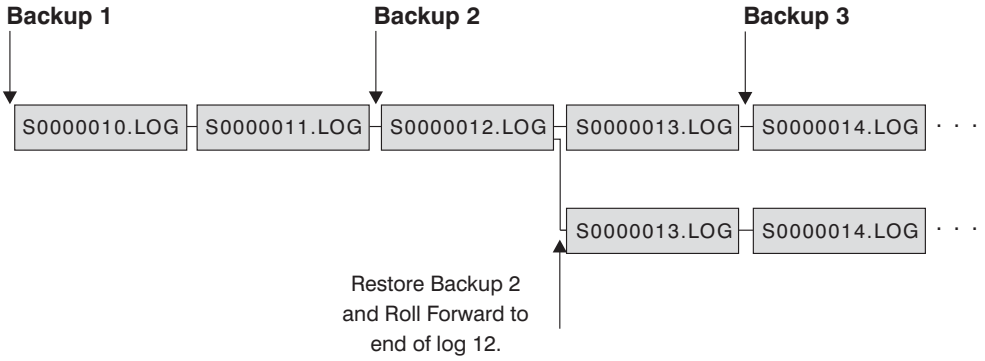


Figure 8. Re-using Log File Names

Related concepts:

- "Log file management through log archiving" on page 49

Related reference:

- Appendix I, "User exit for database recovery," on page 409

Log file allocation and removal

Log files in the database log directory are never removed if they might be required for crash recovery. However, if you have enabled infinite logging, log files will be deleted once they have been successfully archived. When the *logarchmeth1* database configuration parameter is not set to `OFF`, 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 the *logarchmeth1* database configuration parameter:

Logarchmeth1 **and** *Logarchmeth2* **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. Secondary log files are deleted when the database is deactivated or when the space they are using is required for the active log files.

Logarchmeth1 **is set to `LOGRETAIN`**
: Archive logging is used. The database is a recoverable database. Both rollforward recovery and crash recovery are enabled. After you archive the log files, you must delete them from the active log path so that the disk space can be reused for new log files. 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.

Logarchmeth1 **is set to a value other than `OFF` or `LOGRETAIN`**
: Archive logging is used. The database is a recoverable database. Both rollforward recovery and crash recovery are enabled. When a log file becomes full, it is automatically archived.

  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 occurs when log files are being archived, archiving is suspended for the amount of time specified by the ARCHRETRYDELAY database configuration parameter. You can also use the NUMARCHRETRY database configuration parameter to specify the number of times that DB2 is to try archiving a log file to the primary or secondary archive directory before it tries to archive log files to the failover directory (specified by the

FAILARCHPATH database configuration parameter). NUMARCHRETRY is only used if the FAILARCHPATH database configuration parameter is set. If NUMARCHRETRY is set to 0, DB2 will continuously retry archiving from the primary or the secondary log path.

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.

Related concepts:

• "Log file management" on page 46

Log file management through log archiving

The following are general considerations that apply to all methods of log archiving:

- Specifying a value for the database configuration parameter *logarchmeth1* indicates that you want the database manager to archive files or to retrieve log files during rollforward recovery of databases using the method specified. 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.
- Locally attached tape drives should not be used to store log files if you are using any of the following:
  - infinite logging
  - online table space level recovery
  - replication
  - the Asynchronous Read Log API (db2ReadLog)
  - high availability disaster recovery (HADR)

  Any of these events can cause a log file to be retrieved, which can conflict with log archiving operations.
- If you are using log archiving, the log manager will attempt to archive active logs as they are filled. In some cases, if a database is deactivated before the log

manager is able to record the archive as successful, the log manager might try to archive the log again when the database is activated. Thus, a log file can be archived more than once.

- When archiving, a log file is passed to the log manager 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 log manager is retained in the log path directory until it is no longer needed for normal processing. At this point, the disk space is reused.
- When a log file has been archived and it contains no open transactions, 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* retrieve log files during crash recovery or rollback unless the *logsecond* database configuration parameter is set to -1.
- Configuring log archiving does not guarantee rollforward recovery to the point of failure, but only attempts to make the failure window smaller. As log files fill, the log manager will asynchronously archive the logs. 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.

  In the case of a failure of the disk or device on which the log path resides, you can use the MIRRORLOGPATH database configuration parameter to ensure that your logs are written to the secondary path, as long as the disk or device on which the mirror log path is located has not also failed.
- The configured size of each individual log file has a direct bearing on log archiving. If each log file is very large, a large amount of data can be lost if a disk fails. If you configure your database to use small log files the log manager will archive the logs more frequently.

  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. Using larger log files is also recommended if archiving each file requires substantial overhead, such as rewinding the tape device or establishing a connection to the archive media.
- If you are using log archiving, the log manager will attempt to archive active logs as they are filled. In some cases, the log manager will archive a log before it is full. This will occur if the log file is truncated either due to database deactivation, issuing of the ARCHIVE LOG command, at the end of an online backup, or issuing the SET WRITE SUSPEND command.

  Note: To free unused log space, the log file is truncated before it is archived.
- If you are archiving logs and backup images to 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 can take place while a backup operation is in progress, an error can occur when the two processes are trying to write to the same tape drive at the same time.

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

- 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.

- If a user exit or vendor 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.
- On Windows operating systems, you cannot use a REXX user exit to archive logs.
- The user exit or vendor 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 or vendor 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, no other database should 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 might 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 database partition that calls the user exit program are open during the rollforward operation, or you can write a user exit program to handle this situation.

Related concepts:

- "Log file management" on page 46

Log archiving using db2tapemgr

Although DB2 does not provide direct archiving of log files to tape devices, if you want to store archived log files to tape you can use the db2tapemgr utility. To use this tool, the source database must have the database configuration parameter LOGARCHMETH1 configured to a disk location on each database partition that must have its log files moved to a tape device. When invoked, the db2tapemgr utility copies archived log files from the disk location specified by LOGARCHMETH1 to a specified tape device and updates the location of the log files in the recovery history file. The configuration parameter LOGARCHMETH2 is not used by the db2tapemgr utility, and can be set to any allowed archive media type.

STORE and DOUBLE STORE options:

Issue the DB2TAPEMGR command with either the STORE or DOUBLE STORE option to transfer archived logs from disk to tape.

- The STORE option stores a range or all log files from the log archive directory to a specified tape device and deletes the files from disk.
- The DOUBLE STORE option scans the history file to see if logs have been stored to tape previously.
  - If a log has never been stored before, DB2TAPEMGR stores the log file to tape and but does not delete it from disk.

– If a log has been stored before, DB2TAPEMGR stores the log file to tape and deletes it from disk.

Use DOUBLE STORE if you want to keep duplicate copies of your archived logs on tape and on disk, or if you want to store the same logs on two different tapes.

When you issue the DB2TAPEMGR command with either the STORE or DOUBLE STORE option, the db2tapemgr utility first scans the history file for entries where the LOGARCHMETH1 configuration parameter is set to disk. If it finds that any files that are supposed to be on disk, are not on disk, it issues a warning. If the db2tapemgr utility finds no log files to store, it stops the operation and issues a message to inform you that there is nothing to do.

If the db2tapemgr utility finds log files on disk, it then reads the tape header to make sure that it can write the log files to the tape. It also updates the history for those files that are currently on tape. If the update fails, the operation stops and an error message is displayed.

If the tape is writeable, the db2tapemgr utility copies the logs to tape. After the files have been copied, the log files are deleted from disk. Finally, the db2tapemgr utility copies the history file to tape and deletes it from disk.

RETRIEVE options:

Issue the DB2TAPEMGR command with the RETRIEVE option to transfer files from tape to disk.

- Use the RETRIEVE ALL LOGS or LOGS n TO n option to retrieve all archived logs that meet your specified criteria and copy them to disk.
- Use the RETRIEVE FOR ROLLFORWARD TO POINT-IN-TIME option to retrieve all archived logs required to perform a rollforward operation and copy them to disk.
- Use the RETRIEVE HISTORY FILE option to retrieve the history file from tape and copy it to disk.

Related concepts:

- "Backing up to tape" on page 68

Related reference:

- "db2tapemgr - Manage log files on tape command" in *Command Reference*

Archiving log files to tape

Use the db2tapemgr utility to store archived log files to tape and to retrieve them from tape

Prerequisites:

You must set the configuration parameter LOGARCHMETH1 to store archived log files to disk in order to use the db2tapemgr utility to move the archived logs to tape.

Restrictions:

- The db2tapemgr utility does not append log files to a tape. If a store operation does not fill the entire tape, then the unused space is wasted.

- The db2tapemgr utility stores log files only once to any given tape. This restriction exists to avoid any problems inherent to writing to tape media, such as stretching of the tape.
- In a partitioned database environment, the db2tapemgr utility only executes against one database partition at a time. You must run the appropriate command for each database partition, specifying the database partition number using the ON DBPARTITIONNUM option of the DB2TAPEMGR command. You must also ensure that each database partition has access to a tape device.

Procedure:

The following example shows how to use the DB2TAPEMGR command to store all log files from the primary archive log path for database sample on database partition number 0 to a tape device and remove them from the archive log path:

```
db2tapemgr db sample on dbpartitionnum 0 store on /dev/rmt0.1 all logs
```

The following example shows how to store the first 10 log files from the primary archive log path to a tape device and remove them from the archive log path:

```
db2tapemgr db sample on dbpartitionnum store on /dev/rmt0.1 10 logs
```

The following example shows how to store the first 10 log files from the primary archive log path to a tape device and then store the same log files to a second tape and remove them from the archive log path:

```
db2tapemgr db sample on dbpartitionnum double store on /dev/rmt0.1 10 logs
db2tapemgr db sample on dbpartitionnum double store on /dev/rmt1.1 10 logs
```

The following example shows how to retrieve all log files from a tape to a directory:

```
db2tapemgr db sample on dbpartitionnum retrieve all logs from /dev/rmt1.1
   to /home/dbuser/archived_logs
```

Related concepts:

- "Log archiving configuration for high availability disaster recovery (HADR)" on page 257
- "Log archiving using db2tapemgr" on page 51

Related reference:

- "Configuration parameters for database logging" on page 37

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 a log archiving method is specified, 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 might 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:

- "Log file management through log archiving" on page 49
- "Understanding recovery logs" on page 33

On demand log archive

DB2 supports the closing (and, if 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 concepts:

- "Log file management through log archiving" on page 49

Related reference:

- "Configuration parameters for database logging" on page 37
- "ARCHIVE LOG " on page 323
- "db2ArchiveLog - Archive the active log file" on page 335

Including log files with a backup image

When performing an online backup operation, you can specify that the log files required to restore and recover a database are included in the backup image. This means that if you need to ship backup images to a disaster recovery site, you do not have to send the log files separately or package them together yourself. Further, you do not have to decide which log files are required to guarantee the consistency of an online backup. This provides some protection against the deletion of log files required for successful recovery.

To make use of this feature specify the INCLUDE LOGS option of the BACKUP DATABASE command. When you specify this option, the backup utility will truncate the currently active log file and copy the necessary set of log extents into the backup image.

To restore the log files from a backup image, use the LOGTARGET option of the RESTORE DATABASE command and specify a fully qualified path that exists on the DB2 server. The restore database utility will then write the log files from the image to the target path. If a log file with the same name already exists in the target path, the restore operation will fail and an error will be returned. If the LOGTARGET option is not specified, no log files will be restored from the backup image.

If the LOGTARGET option is specified and the backup image does not include any log files, an error will be returned before an attempt is made to restore any table space data. The restore operation will also fail if an invalid or read-only path is

specified. During a database or table space restore where the LOGTARGET option is specified, if one or more log files cannot be extracted, the restore operation fails and an error is returned.

You can also choose to restore only the log files saved in the backup image. To do this, specify the LOGS option with the LOGTARGET option of the RESTORE DATABASE command. If the restore operation encounters any problems when restoring log files in this mode, the restore operation fails and an error is returned.

During an automatic incremental restore operation, only the logs included in the target image of the restore operation will be retrieved from the backup image. Any logs that are included in intermediate images referenced during the incremental restore process will not be extracted from those backup images. During a manual incremental restore, if you specify a log target directory when restoring a backup image that includes log files, the log files in that backup image will be restored.

If you roll a database forward that was restored from an online backup image that includes log files, you might encounter error SQL1268N, which indicates roll-forward recovery has stopped due to an error received when retrieving a log. This error is generated when the target system to which you are attempting to restore the backup image does not have access to the facility used by the source system to archive its transaction logs.

If you specify the INCLUDE LOGS option of the BACKUP DATABASE command when you back up a database, then subsequently perform a restore operation and a roll-forward operation that use that backup image, DB2 will still search for additional transaction logs when rolling the database forward, even though the backup image includes logs. It is standard rollforward behaviour to continue to search for additional transaction logs until no more logs are found. It is possible to have more than one log file with the same timestamp. Consequently, DB2 does not stop as soon as it finds the first timestamp that matches the point-in-time to which you are rolling forward the database as there might be other log files that also have that timestamp. Instead, DB2 continues to look at the transaction log until it finds a timestamp greater than the point-in-time specified.

When no additional logs can be found, the rollforward operation ends successfully. However, if there is an error while searching for additional transaction log files, error SQL1268N is returned. Error SQL1268N can occur because during the initial restore, certain database configuration parameters were reset or overwritten. Three of these database configuration parameters are the TSM parameters, TSM_NODENAME, TSM_OWNER and TSM_PASSWORD. They are all reset to NULL. To rollforward to the end of logs, you need to reset these database configuration parameters to correspond to the source system prior to the rollforward operation. Alternatively, you can specify the NORETRIEVE option when you issue the ROLLFORWARD DATABASE command. This will prevent the DB2 database system from trying to obtain potentially missing transaction logs elsewhere.

Notes:

1. This feature is available only on single-partition databases.
2. This feature is not supported for offline backups.
3. When logs are included in an online backup image, the resulting image cannot be restored on releases of DB2 database prior to Version 8.2.

Related tasks:

- "Restoring from incremental backup images" on page 28

Related reference:
- "BACKUP DATABASE " on page 71
- "RESTORE DATABASE " on page 100

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 might 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 might be required for rollforward recovery because some of them might 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 might have to drop the database (or drop the database on one database 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 might 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 might have been archived; however, DB2 might 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 might have to determine which set of logs should be used for future recovery operations.

Related concepts:
- "Understanding recovery logs" on page 33

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 database is automatically rebuilt and more than one image is restored
- 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 (when dropped table recovery is enabled)
- A table is reorganized
- On-demand log archiving is invoked
- A new log file is written to (when using recoverable logging)
- A log file is archived (when using recoverable logging)
- A database is recovered

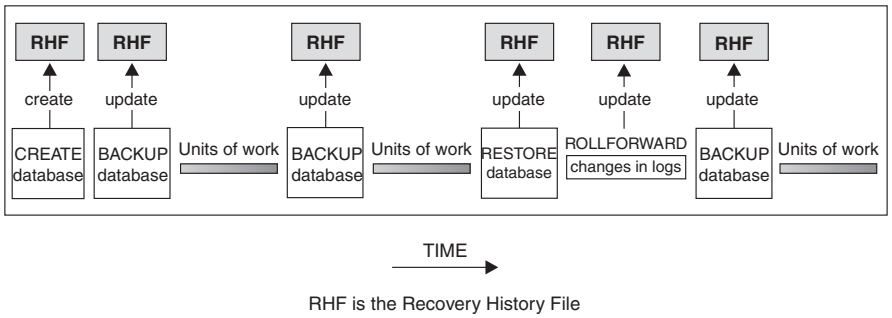


Figure 9. Creating and Updating the Recovery History File

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 associated with 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 recovery history 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:

- "LIST HISTORY " on page 326
- "rec_his_retentn - Recovery history retention period configuration parameter" in *Performance Guide*

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 or full (non-incremental) table space operation completes successfully.
- After a database restore operation, where a rollforward operation is not required, completes successfully.
- After a 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.

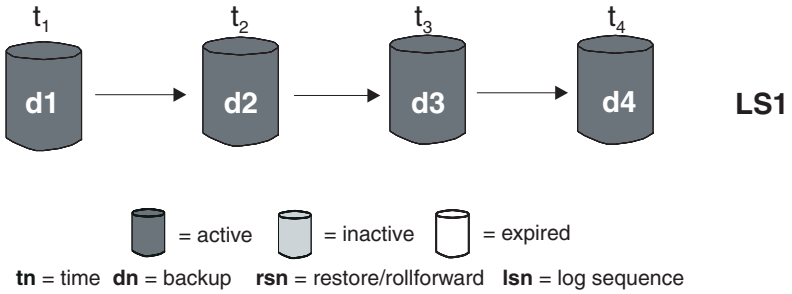


Figure 10. Active Database Backups. The value of *num_db_backups* has been set to four.

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”.

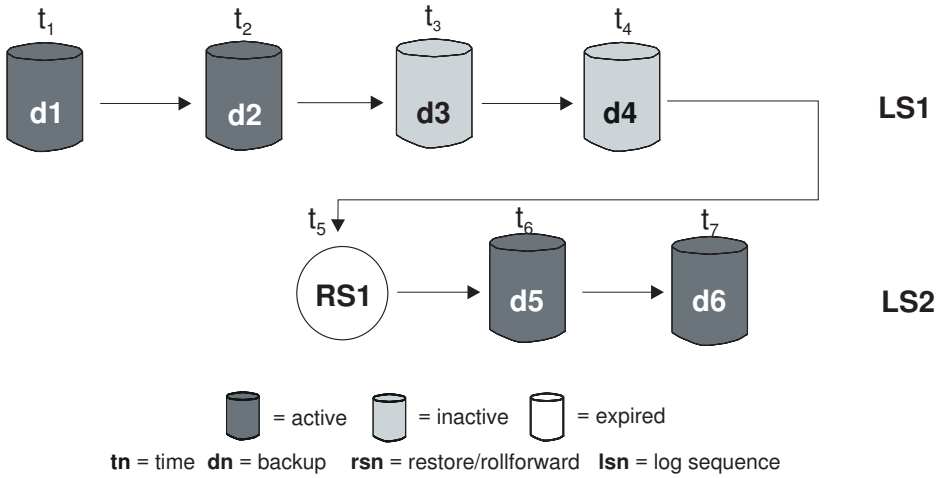


Figure 11. Inactive Database Backups

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”.

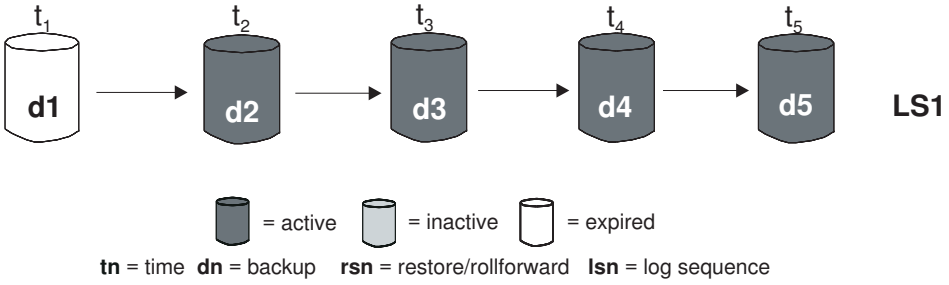


Figure 12. Expired Database Backups

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.)

When a database is migrated, all online database backup entries and all online or offline table space backup entries in the history file are marked as "expired", so that these entries are not selected by automatic rebuild as images required for rebuilding. Load copy images and log archive entries are also marked as "expired", since these types of entries cannot be used for recovery purposes.

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.

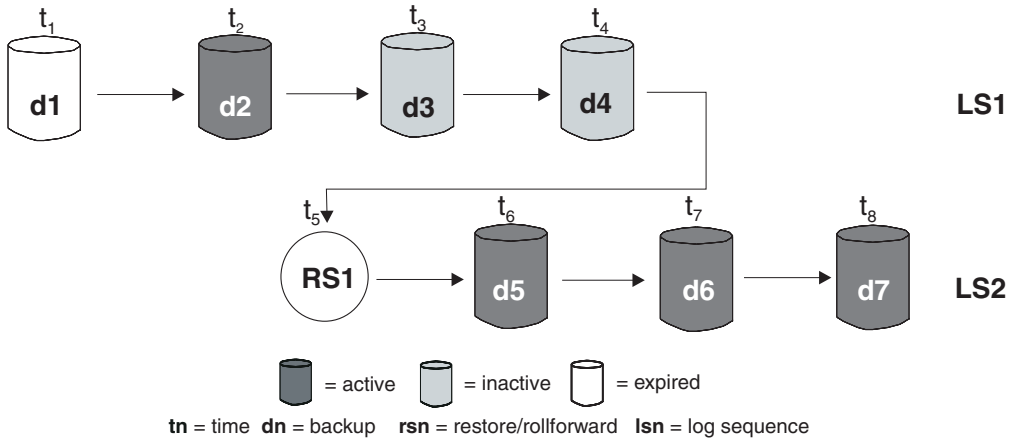


Figure 13. Mixed Active, Inactive, and Expired Database Backups

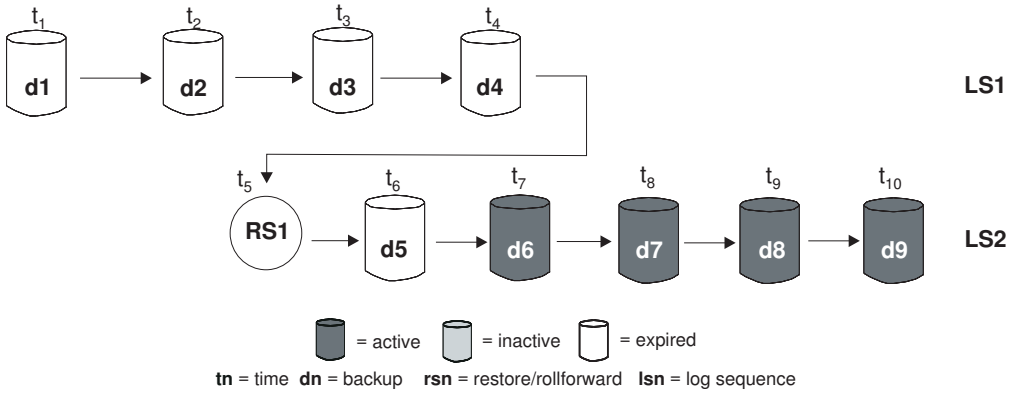


Figure 14. Expired Log Sequence

Related concepts:

- "Understanding the recovery history file" on page 56

Related reference:

- "PRUNE HISTORY/LOGFILE " on page 329

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:

- *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.)

- *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. A table space is also put in this state if, during a restore operation, the table space cannot be restored.
- *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.
- *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.

Related concepts:

- "Recovering damaged table spaces" on page 12

Related tasks:

- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding selected table spaces" on page 139
- "Recovering table spaces in non-recoverable databases" on page 13
- "Recovering table spaces in recoverable databases" on page 12

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.

  DB2 automatically attempts to minimize the time it takes to complete a backup or restore operation by choosing an optimal value for the number of buffers, the buffer size and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available and the database configuration.
- To reduce the amount of time required to complete a restore operation, use multiple 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 devices should be used.
  - Do not overload the I/O device controller bandwidth.
- 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:

- "Optimizing backup performance" on page 84
- "Restore overview" on page 89

Chapter 2. Database backup

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

The following topics are covered:

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.

Note: If the DB2 client and server are not located on the same system, DB2 will determine which directory is the current working directory on the client machine and use that as the backup target directory on the server. For this reason, it is recommended that you specify a target directory for the backup image.

Backup images are created at the target location specified 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.

If you want to delete old backup images that are no longer required, you can remove the files if the backups are stored as files. If you subsequently run a LIST HISTORY command with the BACKUP option, information about the deleted backup images will also be returned. You must use the PRUNE command to remove those entries from the recovery history file.

If your backup objects were saved using Tivoli Storage Manager (TSM), you can use the db2adutl utility to query, extract, verify, and delete the backup objects. On Linux and UNIX, this utility is located in the sqllib/adsm directory, and on Windows operating systems, it is located in sqllib\bin.

On all operating 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
```

Note: DB2 Universal Database™, Version 8.2.2 and earlier versions used a four-level subdirectory tree when storing backup images on Windows operating systems:

```
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 database partition number. In single partition database environments, this is always `NODE0000`. In partitioned database environments, it is `NODE`*xxxx*, where *xxxx* is the number assigned to the database partition in the `db2nodes.cfg` file.

Catalog partition number
The database partition number of the catalog partition for the database. In single partition database environments, this is always `CATN0000`. In partitioned database environments, it is `CATN`*xxxx*, where *xxxx* is the number assigned to the database partition 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. In a large partitioned database environment, however, it might 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.
- In a partitioned database environment, 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.

Concurrent backup operations on the same table space are not permitted. Once a backup operation has been initiated on a table space, any subsequent attempts will fail (SQL2048).

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 might 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:

- "Automatic database backup" on page 85
- "Developing a backup and recovery strategy" on page 3
- "Including log files with a backup image" on page 54
- "Understanding the recovery history file" on page 56

Related reference:

- Appendix G, "Tivoli Storage Manager," on page 403

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.

Related reference:

- "BACKUP DATABASE " on page 71
- "db2Backup - Back up a database or table space" on page 76

Using backup

Use the BACKUP DATABASE command to take a copy of a database's data and store it on a different medium in case of failure or damage to the original. You can back up an entire database, database partition, or only selected table spaces.

Prerequisites:

You do not need to be connected to the database that is to be backed up: the backup database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the backup operation. If you are connected to a database that is to be backed up, you will be disconnected when the BACKUP DATABASE command is issued and the backup operation will proceed.

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).

If you are performing an offline backup and if you have activated the database using the ACTIVATE DATABASE statement, you must deactivate the database before you run the offline backup. If there are active connections to the database, in order to deactivate the database successfully, a user with SYSADM authority must connect to the database and issue the following commands:

```
CONNECT TO database-alias
QUIESCE DATABASE immediate
CONNECT RESET
DEACTIVATE DATABASE database-alias
```

In a partitioned database environment, 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 partition first, then back up the other database partitions. You can also use the Command Editor 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.
- Online backup operations for DMS table spaces are incompatible with the following operations:
  - load
  - reorganization (online and offline)
  - drop table space
  - table truncation
  - index creation
  - not logged initially (used with the CREATE TABLE and ALTER TABLE statements)

- If you attempt to perform an offline backup of a database that is currently active, you will receive an error. Before you run an offline backup you can make sure the database is not active by issuing the DEACTIVATE DATABASE command.

Procedure:

The backup utility can be invoked through the command line processor (CLP), the Backup Database 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 wizard:

1. From the Control Center, expand the object tree until you find the database or table space object that you want to back up.
2. Right-click on the object and select Backup from the pop-up menu. The Backup Database wizard opens.

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

If you performed an offline backup, after the backup completes, you must reactivate the database:

```
ACTIVATE DATABASE  database-alias
```

Related concepts:

- "Administrative APIs in Embedded SQL or DB2 CLI Programs" in *Administrative API Reference*
- "Introducing the plug-in architecture for the Control Center" in *Administration Guide: Implementation*

Related tasks:

- "Migrating databases" in *Migration Guide*

Related reference:

- "BACKUP DATABASE " on page 71
- "db2Backup - Back up a database or table space" on page 76
- "DEACTIVATE DATABASE command" in *Command Reference*
- "LIST DBPARTITIONNUMS command" in *Command Reference*

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 might 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 might take a long time to complete. You might 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 might return an error. If this happens, you might need to rewrite the image using an appropriate block size. Following is an example on AIX:

```
tctl -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_filename.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 might 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	61 440	15
3490	s370	61 440	15
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.

Related concepts:

- "Backup overview" on page 63

Related reference:

- "BACKUP DATABASE " on page 71
- "db2Backup - Back up a database or table space" on page 76

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. 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
```

3. Use this pipe as the target for a database backup operation:

```
db2 backup db sample to /u/dmcinnis/mypipe
```

Related tasks:

- "Using backup" on page 66

Related reference:

- "BACKUP DATABASE " on page 71
- "RESTORE DATABASE " on page 100

BACKUP DATABASE

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

For information on the backup operations supported by DB2 database systems between different operating systems and hardware platforms, see "Backup and restore operations between different operating systems and hardware platforms" in the *Related concepts* section.

Scope:

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

Authorization:

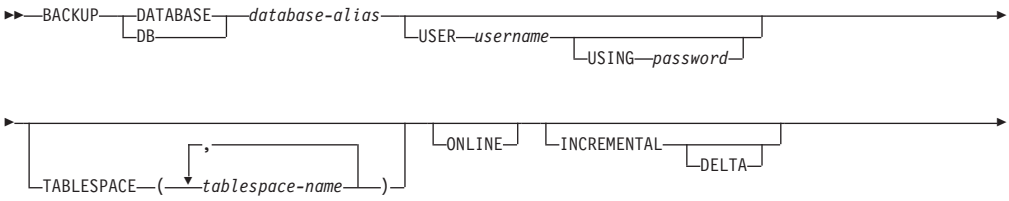
One of the following:

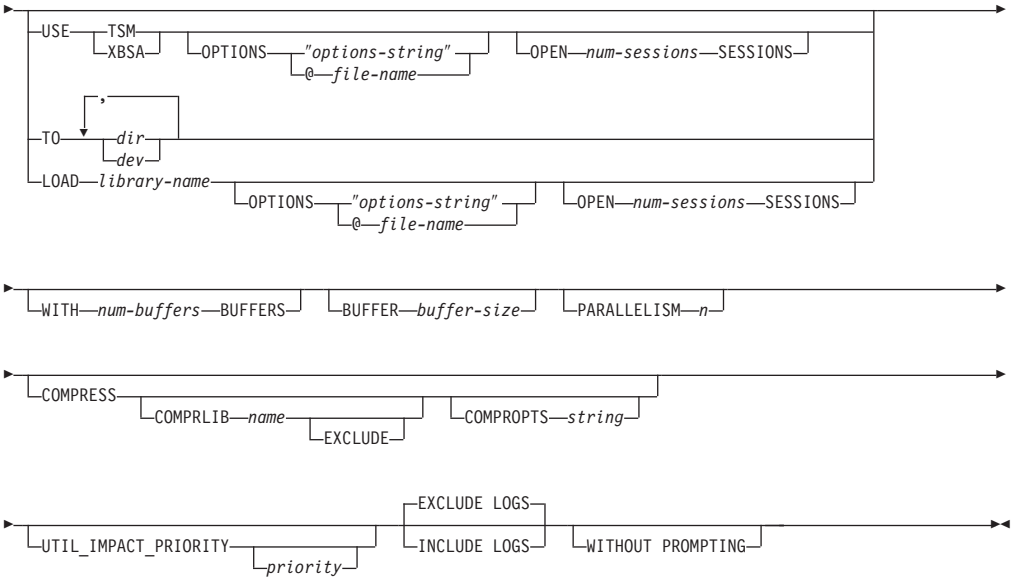
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

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





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. During an online backup, DB2 obtains IN (Intent None) locks on all tables existing in SMS table spaces as they are processed and S (Share) locks on LOB data in SMS table spaces.

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 output.

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.

OPTIONS

"options-string"
Specifies options to be used for the backup operation.The string will be passed to the vendor support library, for example TSM, exactly as it was entered, without the quotes. Specifying this option overrides the value specified by the VENDOROPT database configuration parameter.

@file-name
Specifies that the options to be used for the backup operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library, for example TSM. The file must be a fully qualified file name.

OPEN num-sessions SESSIONS
The number of I/O sessions to be created between DB2 and TSM or another backup vendor product. This parameter has no effect when backing up to tape, disk, or other local device.

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 can 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 might 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. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. However, when creating a backup to multiple locations, a larger number of buffers can be used to improve performance.

BUFFER buffer-size
The size, in 4 KB pages, of the buffer used when building the backup image. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

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

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 you want to use with the BUFFER parameter, 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. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value.

UTIL_IMPACT_PRIORITY *priority*
Specifies that the backup will run in throttled mode, with the priority specified. Throttling allows you to regulate the performance impact of the backup operation. Priority can be any number between 1 and 100, with 1 representing the lowest priority, and 100 representing the highest priority. If the UTIL_IMPACT_PRIORITY keyword is specified with no priority, the backup will run with the default priority of 50. If UTIL_IMPACT_PRIORITY is not specified, the backup will run in unthrottled mode. An impact policy must be defined by setting the *util_impact_lim* configuration parameter for a backup to run in throttled mode.

COMPRESS
Indicates that the backup is to be compressed.

COMPRLIB *name*
Indicates the name of the library to be used to perform the compression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, the default DB2 compression library will be used. If the specified library cannot be loaded, the backup will fail.

EXCLUDE
Indicates that the compression library will not be stored in the backup image.

COMPROPTS *string*
Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents

of string with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for *string* is 1024 bytes.

EXCLUDE LOGS
Specifies that the backup image should not include any log files. When performing an offline backup operation, logs are excluded whether or not this option is specified.

INCLUDE LOGS
Specifies that the backup image should include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup.

WITHOUT PROMPTING

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

Examples:

1. In the following example, the database WSDB is defined on all 4 database partitions, numbered 0 through 3. The path /dev3/backup is accessible from all database partitions. Database 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 database partition 0). All four database partition backup images will be stored in the /dev3/backup directory.

2. In the following example database SAMPLE is backed up to a TSM server using two concurrent TSM client sessions. DB2 calculates the optimal buffer size for this environment.

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

3. In the following example, a table space-level backup of table spaces (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
   ```

4. The USE TSM OPTIONS keywords can be used to specify the TSM information to use for the backup operation. The following example shows how to use the USE TSM OPTIONS keywords to specify a fully qualified file name:

   ```
   db2 backup db sample use TSM options @/u/dmcinnis/myoptions.txt
   ```

   The file myoptions.txt contains the following information: -fromnode=bar -fromowner=dmcinnis

5. 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
```

6. In the following example, three identical target directories are specified for a backup operation on database SAMPLE. You might want to do this if the target file system is made up of multiple physical disks.

   ```
   db2 backup database sample to /dev3/backup, /dev3/backup, /dev3/backup
   ```

   The data will be concurrently backed up to the three target directories, and three backup images will be generated with extensions `.001`, `.002`, and `.003`.

Usage notes:

The data in a backup cannot be protected by the database server. Make sure that backups are properly safeguarded, particularly if the backup contains LBAC-protected data.

When backing up to tape, 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.

Related concepts:

- "Backup and restore operations between different operating systems and hardware platforms" on page 9
- "Developing a backup and recovery strategy" on page 3

Related tasks:

- "Using backup" on page 66

Related reference:

- "BACKUP DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

db2Backup - Back up a database or table space

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
```

API and data structure syntax:

```
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;
  db2Uint32 iUtilImpactPriority;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2BackupStruct;

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

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

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;
  db2Uint32 iUtilImpactPriority;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} 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;
```

db2Backup 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.

db2BackupStruct data structure parameters:

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

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.

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

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 parameter. The valid values for locationType parameter (defined in sqlutil header file, located in the include directory) 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 the db2MediaListStruct structure.

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

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

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 header file, located in the include directory) 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 4 KB allocation units (pages). Minimum is 8 units.

iNumBuffers
Input. Specifies number of backup buffers to be used. Minimum is 2. Maximum is limited by memory.

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

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 header file, located in the include directory) 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.

DB2BACKUP_COMPRESS
Specifies that the backup should be compressed.

DB2BACKUP_INCLUDE_COMPR_LIB
Specifies that the library used for compressing the backup should be included in the backup image.

DB2BACKUP_EXCLUDE_COMPR_LIB
Specifies that the library used for compressing the backup should be not included in the backup image.

DB2BACKUP_INCLUDE_LOGS
Specifies that the backup image should also include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup or a multi-partition backup.

DB2BACKUP_EXCLUDE_LOGS
Specifies that the backup image should not include any log files.

Note: When performing an offline backup operation, logs are excluded whether or not this option is specified.

iUtilImpactPriority
Input. Specifies the priority value to be used during a backup.

- If this value is non-zero, the utility will run throttled. Otherwise, the utility will run unthrottled.
- If there are multiple concurrent utilities running, this parameter is used to determine a relative priority between the throttled tasks. For example, consider two concurrent backups, one with priority 2 and another with priority 4. Both will be throttled, but the one with priority 4 will be allotted more resources. Setting priorities to 2 and 4 is no different than setting them to 5 and 10 or 30 and 60. Priorities values are purely relative.

piComprLibrary
Input. Indicates the name of the external library to be used to perform compression of the backup image. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will use the default library for compression. If the specified library is not found, the backup will fail.

piComprOptions
Input. Describes a block of binary data that will be passed to the

initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize
Input. A four-byte unsigned integer representing the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

db2TablespaceStruct data structure specific parameters:

tablespaces
Input. 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
Input. Number of entries in the tablespaces parameter.

db2MediaListStruct data structure parameters:

locations
Input. 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
Input. The number of entries in the locations parameter.

locationType
Input. A character indicating the media type. Valid values (defined in sqlutil header file, located in the include directory.) are:

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

SQLI_XBSA_MEDIA
XBSA interface.

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).

db2gBackupStruct data structure specific parameters:

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

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.h).

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.h).

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.

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

iComprLibraryLen
Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in piComprLibrary. Set to zero if no library name is given.

db2Char data structure parameters:

pioData
A pointer to a character data buffer. If NULL, no data will be returned.

iLength
Input. The size of the pioData buffer.

oLength
Output. The number of valid characters of data in the pioData buffer.

Usage notes:

This function is exempt from all label-based access control (LBAC) rules. It backs up all data, even protected data. Also, the data in the backup itself is not protected by LBAC. Any user with the backup and a place in which to restore it can gain access to the data.

Related tasks:

- "Using backup" on page 66

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "BACKUP DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "BACKUP DATABASE " on page 71
- "db2Rollforward - Roll forward a database" on page 177
- "db2Restore - Restore a database or table space" on page 115
- "db2Recover - Restore and roll forward a database" on page 199

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 compute the optimal number of buffers. The optimal size of the buffers, in 4 KB pages, is automatically calculated based on the amount of memory and the number of target devices that are available. The parallelism setting is also automatically calculated and is based on the number or processors available and the number of table spaces to be backed up.

```
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 66

Optimizing backup performance

When you perform a backup operation, DB2 will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available, and the database configuration. The objective is to minimize the time it takes to complete a backup operation. Unless you explicitly enter a value for the following BACKUP DATABASE command parameters, DB2 will select one for them:

- WITH num-buffers BUFFERS
- PARALLELISM n
- BUFFER buffer-size

If the number of buffers and the buffer size are not specified, resulting in DB2 setting the values, it should have minimal affect on large databases. However, for small databases, it can cause a large percentage increase in backup image size. Even if the last data buffer written to disk contains little data, the full buffer is

written to the image anyway. In a small database, this means that a considerable percentage of the image size might be empty.

You can also choose to do any of the following 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 to read data from the database and to compress data during a compressed backup operation. Each process or thread is assigned to a specific table space, so there is no benefit to specifying a value for the PARALLELISM parameter that is larger than the number of table spaces being backed up. When it finishes backing up this table space, it requests another. Note, however, that each process or thread requires both memory and CPU overhead.
- Increase the backup buffer size.

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

  Use at least twice as many buffers as backup targets (or sessions) to ensure that the backup target devices do not have to wait for data.
- Use multiple target devices.

Related concepts:

- "Backup overview" on page 63

Related tasks:

- "Using backup" on page 66

Automatic database backup

A database may become unusable due to a wide variety of hardware or software failures. Automatic database backup simplifies database backup management tasks for the DBA by always ensuring that a recent full backup of the database is performed as needed. It determines the need to perform a backup operation based on one or more of the following measures:

- You have never completed a full database backup
- The time elapsed since the last full backup is more than a specified number of hours
- The transaction log space consumed since the last backup is more than a specified number of 4 KB pages (in archive logging mode only).

Protect your data by planning and implementing a disaster recovery strategy for your system. If suitable to your needs, you may incorporate the automatic database backup feature as part of your backup and recovery strategy.

If the database is enabled for roll-forward recovery (archive logging), then automatic database backup can be enabled for either online or offline backup. Otherwise, only offline backup is available. Automatic database backup supports disk, tape, Tivoli Storage Manager (TSM), and vendor DLL media types.

Through the Configure Automatic Maintenance wizard in the Control Center or Health Center, you can configure:

- The requested time or number of log pages between backups
- The backup media
- Whether it will be an online or offline backup.

If backup to disk is selected, the automatic backup feature will regularly delete backup images from the directory specified in the Configure Automatic Maintenance wizard. Only the most recent backup image is guaranteed to be available at any given time. It is recommended that this directory be kept exclusively for the automatic backup feature and not be used to store other backup images.

The automatic database backup feature can be enabled or disabled by using the **auto_db_backup** and **auto_maint** database configuration parameters. In a partitioned database environment, the automatic database backup runs on each database partition if the database configuration parameters are enabled on that database partition.

Related concepts:

- "Developing a backup and recovery strategy" on page 3
- "Automatic statistics collection" in *Performance Guide*
- "Catalog statistics" in *Performance Guide*
- "Table and index management for MDC tables" in *Performance Guide*
- "Table and index management for standard tables" in *Performance Guide*
- "Table reorganization" in *Performance Guide*
- "Health monitor" in *System Monitor Guide and Reference*

Related reference:

- "auto_maint - Automatic maintenance configuration parameter" in *Performance Guide*

Enabling automatic backup

A database can become unusable due to a wide variety of hardware or software failures. Ensuring that you have a recent, full backup of your database is an integral part of planning and implementing a disaster recovery strategy for your system. Use automatic database backup as part of your disaster recovery strategy to enable DB2 to back up your database both properly and regularly.

Procedure:

You can turn this feature on using either the graphical user interface tools or the command line interface.

- To set up your database for automatic backup using the graphical user interface tools:

1. Open the Configure Automatic Maintenance wizard either from the Control Center by right-clicking on a database object or from the Health Center by right-clicking on the database instance that you want to configure for automatic backup. Select **Configure Automatic Maintenance** from the pop-up window.
2. Within this wizard, you can enable automatic backup and specify a maintenance window for the execution of the BACKUP utility.

- To set up your database for automatic backup using the command line interface, set each of the following configuration parameters to ON:
  - AUTO_MAINT
  - AUTO_DB_BACKUP

Related concepts:

- "Automatic database backup" on page 85

Compatibility of online backup and other utilities

Some utilities can be run at the same time as an online backup, but others cannot.

The following utilities are compatible with online backup:

- EXPORT
- ONLINE INSPECT

The following utilities are compatible with online backup only under certain circumstances:

- ONLINE CREATE INDEX

  In SMS mode, online index create and online backup will not be compatible due to the ALTER TABLE lock. Online index create acquires it in exclusive mode while online backup acquires it in share.

  In DMS mode, online index create and online backup can run concurrently in most cases. There is a possibility if you have a large number of indexes that the online index create will internally acquire an online backup lock that will conflict with any concurrent online backup.
- ONLINE INDEX REORG

  As with online index create, online index reorganization is not compatible with online backup due to the ALTER TABLE lock. Online index reorganization acquires it in exclusive mode while online backup acquires it in share.

  In DMS mode, online index reorganization and online backup can run concurrently in most cases. As with online index create, there is a possibility if you have a large number of indexes that the online index reorganization will internally acquire an online backup lock that will conflict with any concurrent online backup.

  In addition, online index reorganization quiesces the table before the switch phase and gets a Z lock, which prevents an online backup.
- REBALANCE

  When online backup and rebalancer are running concurrently, online backup will pause the rebalancer and does not wait for it to complete.
- IMPORT

  The import utility is compatible with online backup except when the IMPORT command is issued with the REPLACE option, in which case, import gets a Z lock on the table and prevents an online backup from running concurrently.

- ONLINE LOAD

  Online load is not compatible with online backup when the LOAD command is issued with the COPY NO option. In this mode the utilities both modify the table space state, causing one of the utilities to report an error.

  Online load is compatible with online backup when the LOAD command is issued with the COPY YES option, although there might still be some compatibility issues. In SMS mode, the utilities can execute concurrently, but they will hold incompatible table lock modes and consequently might be subject to table lock waits. In DMS mode, the utilities both hold incompatible "Internal-B" (OLB) lock modes and might be subject to waits on that lock. If the utilities execute on the same table space concurrently, the load utility might be forced to wait for the backup utility to complete processing of the table space before the load utility can proceed.
- ONLINE TABLE REORG

  The clean up phase of online table reorganization cannot start while an online backup is running. You can pause the table reorganization, if required, to allow the online backup to finish before resuming the online table reorg.

  You can start an online backup of a DMS table space when a table within the same table space is being reorganized online. There might be lock waits associated with the reorganization operation during the truncate phase.

  You cannot start an online backup of an SMS table space when a table within the same table space is being reorganized online. Both operations require an exclusive lock.
- DDLs that require a Z lock (such as ALTER TABLE, DROP TABLE and DROP INDEX)

  Online DMS table space backup is compatible with DDLs that require a Z lock.

  Online SMS table space backup must wait for the Z lock to be released.

The following utilities are not compatible with online backup:

- REORG TABLE
- RESTORE
- ROLLFORWARD
- RUNSTATS
- ONLINE BACKUP
- OFFLINE LOAD
- SET WRITE

Related concepts:

- "Backup overview" on page 63

Chapter 3. Database restore

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

The following topics are covered:

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 and will be replaced when the RESTORE DATABASE command is issued, 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`, 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.

Related concepts:

- "Database rebuild" on page 130
- "Including log files with a backup image" on page 54

Related tasks:

- "Using restore" on page 90

Related reference:

- "db2Restore - Restore a database or table space" on page 115
- "Restore sessions - CLP examples" on page 127

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.

Related reference:

- "db2Restore - Restore a database or table space" on page 115
- "RESTORE DATABASE " on page 100

Using restore

Use the RESTORE DATABASE command to recover a database or table space after a problem such as media or storage failure, power interruption, or application failure. If you have backed up your database, or individual table spaces, you can recreate them if they have become damaged or corrupted in some way.

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. Restore will overwrite the code page of the destination database with that of the backup image.

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 into an existing database 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. The database on disk and in the backup image must be the same.
- You cannot issue a table space-level restore of a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.
- You cannot restore a backup taken in a single database partition environment into an existing partitioned database environment. Instead you must restore the backup to a single database partition environment and then add database partitions as required.
- When restoring a backup image with one code page into a system with a different codepage, the system code page will be overwritten by the code page of the backup image.
- You cannot use the RESTORE DATABASE command to convert non-automatic storage enabled table spaces to automatic storage enabled table space.

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 wizard:

1. From the Control Center, expand the object tree until you find the database or table space object that you want to restore.
2. Right-click on the object and select Restore from the pop-up menu. The Restore wizard opens.

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

Related concepts:

- "Introducing the plug-in architecture for the Control Center" in *Administration Guide: Implementation*
- "Administrative APIs in Embedded SQL or DB2 CLI Programs" in *Administrative API Reference*
- "Database rebuild" on page 130
- "Restore overview" on page 89

Related reference:

- "db2Restore - Restore a database or table space" on page 115
- "Restore sessions - CLP examples" on page 127
- "RESTORE DATABASE " on page 100

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 27

Related reference:

- "BACKUP DATABASE " on page 71
- "LIST HISTORY " on page 326
- "RESTORE DATABASE " on page 100

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 Restore Database wizard in the Control Center. The process for invoking a redirected restore of an incremental backup image is similar to the process for invoking a redirected restore of a non-incremental backup image. Issue the RESTORE DATABASE command with the REDIRECT option and specify the backup image that should be used for the incrementally restore of the database. Alternatively, you can have DB2 generate a redirected restore script from a backup image, then you can modify the script as required. See Performing a redirected restore using an automatically generated script.

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.

The following example shows how to perform a redirected restore on database SAMPLE:

```
db2 restore db sample redirect without prompting
SQL1277W A redirected restore operation is being performed.
Table space configuration can now be viewed and table spaces that do not
use automatic storage can have their containers reconfigured.

DB20000I The RESTORE DATABASE command completed successfully.

db2 set tablespace containers for 2 using (path 'userspace1.0', path
'userspace1.1')
```

```
DB20000I The SET TABLESPACE CONTAINERS command completed successfully.

db2 restore db sample continue
DB20000I The RESTORE DATABASE command completed successfully.
```

Related concepts:

- "Redefine table space containers by restoring a database using an automatically generated script" on page 97

Related tasks:

- "Performing a redirected restore using an automatically generated script" on page 99

Related reference:

- "Restore sessions - CLP examples" on page 127

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 any database or table space backup image to an existing database. For a database-level restore, the backup image can differ from the existing database in its alias name, its database name, or its database seed. For a table-space level restore, you can restore a table space into an existing database 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. The database on disk and in the backup image must be the same.

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 or contains no entries, the database manager copies the file from the backup image. If you want to replace the recovery history file you can issue the RESTORE command with the REPLACE HISTORY FILE option.
- 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.

Related concepts:

• "Restore overview" on page 89

Related tasks:

• "Using restore" on page 90

Related reference:

• "db2Restore - Restore a database or table space" on page 115
• "RESTORE DATABASE " on page 100

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.
• Overwrites the code page of the database with the codepage of the backup image.

Related tasks:

• "Using restore" on page 90

Related reference:

• "db2Restore - Restore a database or table space" on page 115
• "RESTORE DATABASE " on page 100

Redefine table space containers by restoring a database using an automatically generated script

When you restore a database, the restore utility assumes that the physical container layout will be identical to that of the database when it was backed up. If you need to change the location or size of any of the physical containers, you must issue the RESTORE DATABASE command with the REDIRECT option. Using this option requires that you specify the locations of physical containers stored in the backup image and provide the complete set of containers for each non-automatic table space that will be altered. You can capture the container information at the time of the backup, but this can be cumbersome.

To make it easier to perform a redirected restore, the restore utility allows you to generate a redirected restore script from an existing backup image by issuing the RESTORE DATABASE command with the REDIRECT option and the GENERATE SCRIPT option. The restore utility examines the backup image, extracts container information from the backup image, and generates a CLP script that includes all of the detailed container information. You can then modify any of the paths or container sizes in the script, then run the CLP script to recreate the database with the new set of containers. The script you generate can be used to restore a database even if you only have a backup image and you do not know the layout of the containers. The script is created on the client. Using the script as your basis, you can decide where the restored database will require space for log files and containers and you can change the log file and container paths accordingly.

The generated script consists of four sections:

Initialization

The first section sets command options and specifies the database partitions on which the command will run. The following is an example of the first section:

```
UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM 0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
```

where

- `S ON` specifies that execution of the command should stop if a command error occurs
- `Z ON SAMPLE_NODE0000.out` specifies that output should be directed to a file named `<dbalias>_NODE<dbpartitionnum>.out`
- `V ON` specifies that the current command should be printed to standard output.

  When running the script on a partitioned database environment, it is important to specify the database partition on which the script commands will run.

RESTORE command with the REDIRECT option

The second section starts the RESTORE command and uses the REDIRECT option. This section can use all of the RESTORE command options, except any options that cannot be used in conjunction with the REDIRECT option. The following is an example of the second section:

```
RESTORE DATABASE SAMPLE
-- USER '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050906194027
```

```
-- DBPATH ON '<target-directory>'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
```

Table space definitions

This section contains table space definitions for each table space in the backup image or specified on the command line. There is a section for each table space, consisting of a comment block that contains information about the name, type and size of the table space. The information is provided in the same format as a table space snapshot. You can use the information provided to determine the required size for the table space. For table spaces that were created using automatic storage, this section does not include a SET TABLESPACE CONTAINERS clause. The following is an example of the third section:

```
-- *****************************************************************
-- ** Tablespace name                         = SYSCATSPACE
-- **   Tablespace ID                         = 0
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 5572
-- *****************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- *****************************************************************
-- ** Tablespace name                         = TEMPSPACE1
-- **   Tablespace ID                         = 1
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = System Temporary data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 0
-- *****************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- *****************************************************************
-- ** Tablespace name                         = DMS
-- **   Tablespace ID                         = 2
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
-- **   High water mark (pages)               = 96
```

```
-- *********************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                          1000
, FILE   '/tmp/dms2'                                          1000
);
```

RESTORE command with the CONTINUE option

The final section issues the RESTORE command with the CONTINUE option, to complete the redirected restore. The following is an example of the final section:

```
RESTORE DATABASE SAMPLE CONTINUE;
```

Related concepts:

- "Redefining table space containers during a restore operation (redirected restore)" on page 94

Related tasks:

- "Performing a redirected restore using an automatically generated script" on page 99

Related reference:

- "db2Restore - Restore a database or table space" on page 115
- "RESTORE DATABASE " on page 100

Performing a redirected restore using an automatically generated script

When you perform a redirected restore operation, you need to specify the locations of physical containers stored in the backup image and provide the complete set of containers for each table space that will be altered. Use the following procedure to generate a redirected restore script based on an existing backup image, modify the generated script, then run the script to perform the redirected restore.

Prerequisites:

You can perform a redirected restore only if the database has been previously backed up using the DB2 backup utility.

Restrictions:

- If the database exists, you must be able to connect to it in order to generate the script. Therefore, if the database requires migration or crash recovery this must be done before you attempt to generate a redirected restore script.
- If you are working in a partitioned database environment, and the target database does not exist, you cannot run the command to generate the redirected restore script concurrently on all database partitions. Instead, the command to generate the redirected restore script must be run one database partition at a time, starting from the catalog partition.

  Alternatively, you can first create a dummy database with the same name as your target database. After the dummy database has been created, you can then generate the redirected restore script concurrently on all database partitions.

- Even if you specify the REPLACE EXISTING option when you issue the RESTORE command to generate the script, the REPLACE EXISTING option will appear in the script commented out.
- For security reasons, your password will not appear in the generated script. You need to fill in the password manually.
- You cannot generate a script for redirected restore using the Restore Wizard in the Control Center.

Procedure:

To perform a redirected restore using a script:

1. Use the restore utility to generate a redirected restore script. The restore utility can be invoked through the command line processor (CLP) or the db2Restore application programming interface (API). The following is an example of the RESTORE DATABASE command with the REDIRECT option and the GENERATE SCRIPT option:

   ```
   db2 restore db test from /home/jseifert/backups taken at 20050304090733
         redirect generate script test_node0000.clp
   ```

   This creates a redirected restore script on the client called `test_node0000.clp`.
2. Open the redirected restore script in a text editor to make any modifications that are required. You can modify:
   - Restore options
   - Automatic storage paths
   - Container layout and paths
3. Run the modified redirected restore script. For example:

   ```
   db2 -tvf test_node0000.clp
   ```

Related concepts:

- "Redefine table space containers by restoring a database using an automatically generated script" on page 97
- "Redefining table space containers during a restore operation (redirected restore)" on page 94

Related reference:

- "db2Restore - Restore a database or table space" on page 115
- "RESTORE DATABASE " on page 100

RESTORE DATABASE

The **RESTORE DATABASE** command recreates a damaged or corrupted database that has been backed up using the DB2 backup utility. The restored database is in the same state that it was in when the backup copy was made. This utility can also overwrite a database with a different image or restore the backup copy to a new database.

For information on the restore operations supported by DB2 database systems between different operating systems and hardware platforms, see ″Backup and restore operations between different operating systems and hardware platforms″ in the *Related concepts* section.

The restore utility can also be used to restore backup images that were produced on DB2 UDB Verson 8. 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 its previous state by invoking the rollforward utility after successful completion of a restore operation.

This utility can also restore a table space level backup.

Incremental images and images only capturing differences from the time of the previous capture (called a "delta image") cannot be restored when there is a difference in operating systems or word size (32-bit or 64-bit).

Following a successful restore operation from one environment to a different environment, no incremental or delta backups are allowed until a non-incremental backup is taken. (This is not a limitation following a restore operation within the same environment.)

Even with a successful restore operation from one environment to a different environment, there are some considerations: packages must be rebound before use (using the BIND command, the REBIND command, or the db2rbind utility); SQL procedures must be dropped and re-created; and all external libraries must be rebuilt on the new platform. (These are not considerations when restoring to the same environment.)

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*

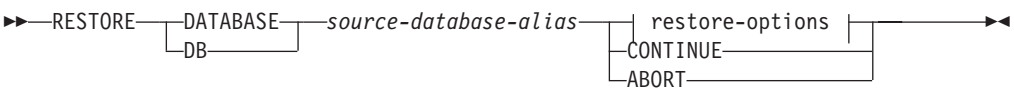
Required connection:

The required connection will vary based on the type of restore action:

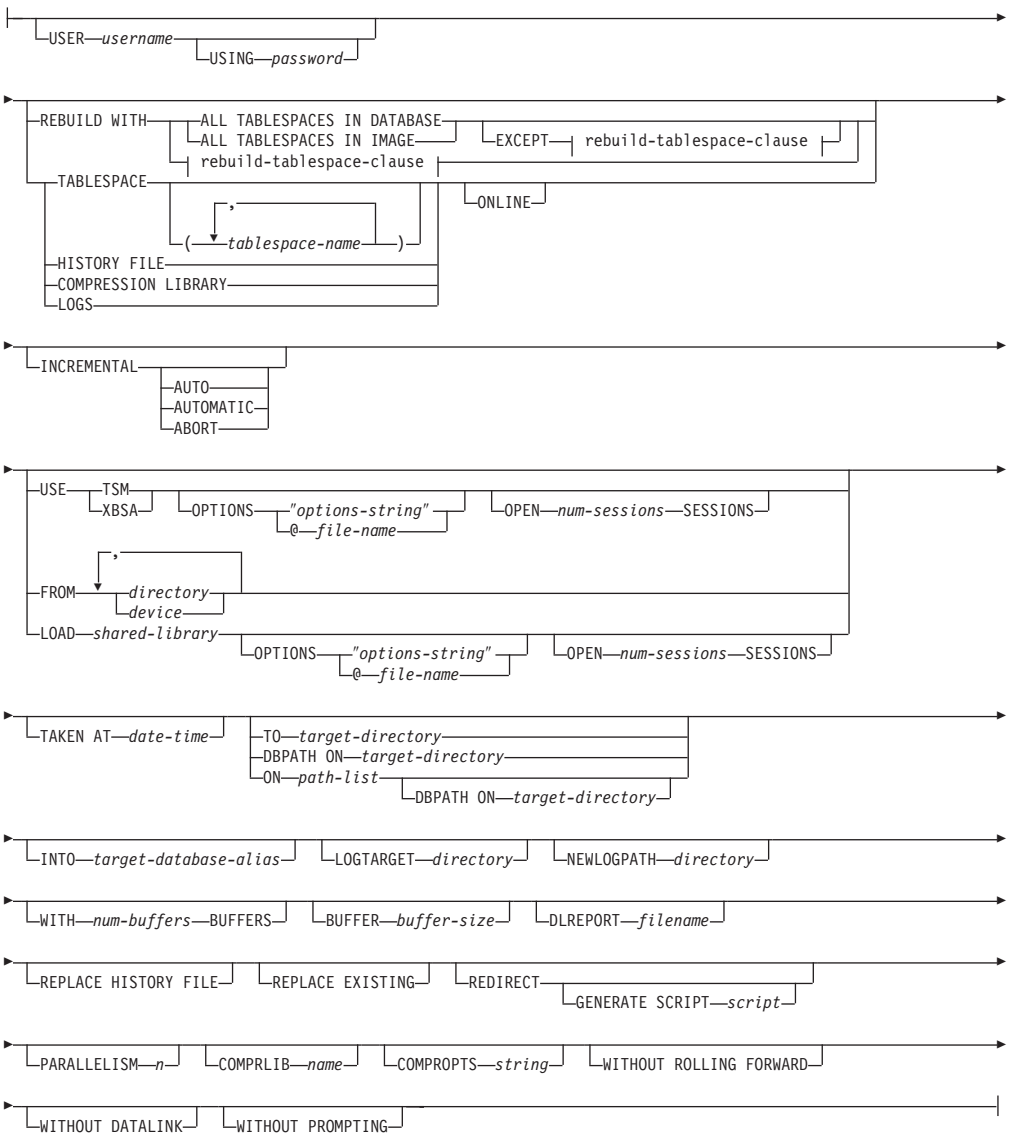
- You require a database connection, to restore to an existing database. This command automatically establishes an exclusive connection to the specified database.
- You require an instance and a database connection, to restore to a new database. The instance attachment is required to create the database.

  To restore to a new database at an instance different from the current instance, it is necessary to first attach to the instance where the new database will reside. The new instance can be local or remote. The current instance is defined by the value of the DB2INSTANCE environment variable.

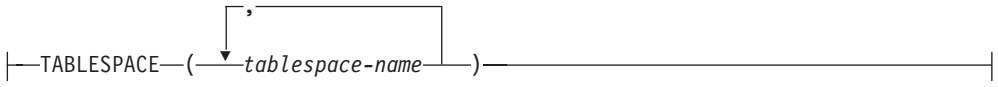
Command syntax:



restore-options:



rebuild-tablespace-clause:



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.

REBUILD WITH ALL TABLESPACES IN DATABASE
Restores the database with all the table spaces known to the database at the time of the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN DATABASE EXCEPT *rebuild-tablespace-clause*
Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE
Restores the database with only the table spaces in the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE EXCEPT *rebuild-tablespace-clause*
Restores the database with only the table spaces in the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH *rebuild-tablespace-clause*
Restores the database with only the list of table spaces specified. This restore overwrites a database if it already exists.

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.

COMPRESSION LIBRARY
This keyword is specified to restore only the compression library from the

backup image. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.

LOGS This keyword is specified to restore only the set of log files contained in the backup image. If the backup image does not contain any log files, the restore operation will fail. If this option is specified, the LOGTARGET option must also be specified.

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.

OPTIONS

"options-string"
Specifies options to be used for the restore operation.The string will be passed to the vendor support library, for example TSM, exactly as it was entered, without the quotes. Specifying this option overrides the value specified by the VENDOROPT database configuration parameter.

@file-name
Specifies that the options to be used for the restore operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library, for example TSM. The file must be a fully qualified file name.

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.

FROM *directory/device*
The fully qualified path name of the directory or device on which the backup image resides. If USE TSM, FROM, and LOAD are omitted, the default value is the current working directory of the client machine. This target directory or device must exist on the target server/instance.

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
```

Using these commands, the DB2 database system generates subdirectories under the `c:\backup` directory to allow more than one backup to be placed in the specified top level directory. The DB2 generated subdirectories should be ignored. To specify precisely which backup image to restore, use the TAKEN AT parameter. There can 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 20021001010101 and 20021002010101 exist, specifying `20021002` causes the image with time stamp `20021002010101` 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*
This parameter states 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. If the backup image contains a database that is enabled for automatic storage then only the database directory changes, the storage paths associated with the database do not change.

DBPATH ON *target-directory*
This parameter states 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. If the backup image contains a database that is enabled for automatic storage and the ON parameter is not specified then this parameter is synonymous with the TO parameter and only the database directory changes, the storage paths associated with the database do not change.

ON *path-list*
This parameter redefines the storage paths associated with an automatic storage database. Using this parameter with a database that is not enabled for automatic storage results in an error (SQL20321N). The existing storage paths as defined within the backup image are no longer used and automatic storage table spaces are automatically redirected to the new

paths. If this parameter is not specified for an automatic storage database then the storage paths remain as they are defined within the backup image.

One or more paths can be specified, each separated by a comma. Each path must have an absolute path name and it must exist locally. If the database does not already exist on disk and the DBPATH ON parameter is not specified then the first path is used as the target database directory.

For a multi-partition database the ON path-list option can only be specified on the catalog partition. The catalog partition must be restored before any other partitions are restored when the ON option is used. The restore of the catalog-partition with new storage paths will place all non-catalog nodes in a RESTORE_PENDING state. The non-catalog nodes can then be restored in parallel without specifying the ON clause in the restore command.

In general, the same storage paths must be used for each partition in a multi-partition database and they must all exist prior to executing the **RESTORE DATABASE** command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You use the argument `" $N" ([blank]$N)` to indicate a database partition expression. A database partition expression can be used anywhere in the storage path, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path after the database partition expression is evaluated. If there is no space character in the storage path after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms:

Table 2. . Operators are evaluated from left to right. `%` represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N	" $N"	10
[blank]$N+[number]	" $N+100"	110
[blank]$N%[number]	" $N%5"	0
[blank]$N+[number]%[number]	" $N+1%5"	1
[blank]$N%[number]+[number]	" $N%4+2"	4
[a] % is modulus.		

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.

LOGTARGET *directory*

The absolute path name of an existing directory on the database server, to be used as the target directory for extracting log files from a backup image. If this option is specified, any log files contained within the backup image will be extracted into the target directory. If this option is not specified, log

files contained within a backup image will not be extracted. To extract only the log files from the backup image, specify the LOGS option.

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 DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. 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 DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

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.

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 HISTORY FILE
Specifies that the restore operation should replace the history file on disk with the history file from the backup image.

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. All commands associated with a single redirected restore operation must be invoked from the same window or CLP session. A redirected restore operation cannot be performed against a table space that has automatic storage enabled.

GENERATE SCRIPT *script*
Creates a redirect restore script with the specified file name. The script

name can be relative or absolute and the script will be generated on the client side. If the file cannot be created on the client side, an error message (SQL9304N) will be returned. If the file already exists, it will be overwritten. Please see the examples below for further usage information.

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.

If this option is specified when restoring from an online backup image, error SQL2537N will be returned.

If backup image is of a recoverable database then WITHOUT ROLLING FORWARD cannot be specified with REBUILD option.

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 created during the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value.

COMPRLIB *name*
Indicates the name of the library to be used to perform the decompression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *string*
Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is "@", the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

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:

1. In the following example, the database WSDB is defined on all 4 database partitions, numbered 0 through 3. The path /dev3/backup is accessible from all database 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. When performing a restore using db2_all, you should always specify REPLACE EXISTING and/or WITHOUT PROMPTING. Otherwise, if there is prompting, the operation will look like it is hanging. This is because db2_all does not support user prompting.

2. Following is a typical redirected restore scenario for a database whose alias is MYDB:
   a. Issue a RESTORE DATABASE command with the REDIRECT option.

      ```
      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:

      ```
      restore db mydb abort
      ```

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

      ```
      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.

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

      ```
      restore db mydb continue
      ```

      This is the final step of the redirected restore operation.

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

4. To produce a backup image, which includes logs, for transportation to a remote site:

   ```
   backup db sample online to /dev3/backup include logs
   ```

   To restore that backup image, supply a LOGTARGET path and specify this path during ROLLFORWARD:

   ```
   restore db sample from /dev3/backup logtarget /dev3/logs
   rollforward db sample to end of logs and stop overflow log path /dev3/logs
   ```

5. To retrieve only the log files from a backup image that includes logs:

   ```
   restore db sample logs from /dev3/backup logtarget /dev3/logs
   ```

6. The USE TSM OPTIONS keywords can be used to specify the TSM information to use for the restore operation. On Windows platforms, omit the -fromowner option.
   - Specifying a delimited string:

     ```
     restore db sample use TSM options '"-fromnode=bar -fromowner=dmcinnis"'
     ```

   - Specifying a fully qualified file:

     ```
     restore db sample use TSM options @/u/dmcinnis/myoptions.txt
     ```

     The file `myoptions.txt` contains the following information: -fromnode=bar -fromowner=dmcinnis

7. The following is a simple restore of a multi-partition automatic storage enabled database with new storage paths. The database was originally created with one storage path, `/myPath0:`
   - On the catalog partition issue: `restore db mydb on /myPath1,/myPath2`
   - On all non-catalog partitions issue: `restore db mydb`

8. A script output of the following command on a non-auto storage database:

   ```
   restore db sample from /home/jseifert/backups taken at 20050301100417 redirect
   generate script SAMPLE_NODE0000.clp
   ```

   would look like this:

   ```
   -- ******************************************************************************
   -- ** automatically created redirect restore script
   -- ******************************************************************************
   UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
   SET CLIENT ATTACH_DBPARTITIONNUM  0;
   SET CLIENT CONNECT_DBPARTITIONNUM 0;
   -- ******************************************************************************
   -- ** initialize redirected restore
   -- ******************************************************************************
   RESTORE DATABASE SAMPLE
   -- USER  '<username>'
   ```

```
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050301100417
-- DBPATH ON '<target-directory>'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- *********************************************************************************
-- ** tablespace definition
-- *********************************************************************************
-- *********************************************************************************
-- ** Tablespace name                          = SYSCATSPACE
-- **   Tablespace ID                          = 0
-- **   Tablespace Type                        = System managed space
-- **   Tablespace Content Type                = Any data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = No
-- **   Total number of pages                  = 5572
-- *********************************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- *********************************************************************************
-- ** Tablespace name                          = TEMPSPACE1
-- **   Tablespace ID                          = 1
-- **   Tablespace Type                        = System managed space
-- **   Tablespace Content Type                = System Temporary data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = No
-- **   Total number of pages                  = 0
-- *********************************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- *********************************************************************************
-- ** Tablespace name                          = USERSPACE1
-- **   Tablespace ID                          = 2
-- **   Tablespace Type                        = System managed space
-- **   Tablespace Content Type                = Any data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = No
-- **   Total number of pages                  = 1
-- *********************************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0002.0'
);
-- *********************************************************************************
-- ** Tablespace name                          = DMS
-- **   Tablespace ID                          = 3
-- **   Tablespace Type                        = Database managed space
```

```
-- **   Tablespace Content Type                 = Any data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 32
-- **   Using automatic storage                 = No
-- **   Auto-resize enabled                     = No
-- **   Total number of pages                   = 2000
-- **   Number of usable pages                  = 1960
-- **   High water mark (pages)                 = 96
-- *********************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   /tmp/dms1                                                    1000
, FILE   /tmp/dms2                                                    1000
);
-- *********************************************************************************
-- ** Tablespace name                         = RAW
-- **   Tablespace ID                         = 4
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
-- **   High water mark (pages)               = 96
-- *********************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                          1000
, DEVICE '/dev/hdb2'                                          1000
);
-- *********************************************************************************
-- ** start redirect restore
-- *********************************************************************************
RESTORE DATABASE SAMPLE CONTINUE;
-- *********************************************************************************
-- ** end of file
-- *********************************************************************************
```

9. A script output of the following command on an automatic storage database:

```
restore db test from /home/jseifert/backups taken at 20050304090733 redirect
generate script TEST_NODE0000.clp
```

would look like this:

```
-- *********************************************************************************
-- ** automatically created redirect restore script
-- *********************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON TEST_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- *********************************************************************************
-- ** initialize redirected restore
-- *********************************************************************************
RESTORE DATABASE TEST
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050304090733
ON '/home/jseifert'
-- DBPATH ON <target-directory>
INTO TEST
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00002/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
```

```
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- *********************************************************************************
-- ** tablespace definition
-- *********************************************************************************
-- *********************************************************************************
-- ** Tablespace name                          = SYSCATSPACE
-- **   Tablespace ID                          = 0
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type                = Any data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 4
-- **   Using automatic storage                = Yes
-- **   Auto-resize enabled                    = Yes
-- **   Total number of pages                  = 6144
-- **   Number of usable pages                 = 6140
-- **   High water mark (pages)                = 5968
-- *********************************************************************************
-- *********************************************************************************
-- ** Tablespace name                          = TEMPSPACE1
-- **   Tablespace ID                          = 1
-- **   Tablespace Type                        = System managed space
-- **   Tablespace Content Type                = System Temporary data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = Yes
-- **   Total number of pages                  = 0
-- *********************************************************************************
-- *********************************************************************************
-- ** Tablespace name                          = USERSPACE1
-- **   Tablespace ID                          = 2
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type                = Any data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = Yes
-- **   Auto-resize enabled                    = Yes
-- **   Total number of pages                  = 256
-- **   Number of usable pages                 = 224
-- **   High water mark (pages)                = 96
-- *********************************************************************************
-- *********************************************************************************
-- ** Tablespace name                          = DMS
-- **   Tablespace ID                          = 3
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type                = Any data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = No
-- **   Auto-resize enabled                    = No
-- **   Total number of pages                  = 2000
-- **   Number of usable pages                 = 1960
-- **   High water mark (pages)                = 96
-- *********************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                        1000
, FILE   '/tmp/dms2'                                        1000
);
-- *********************************************************************************
```

```
-- ** Tablespace name                          = RAW
-- **   Tablespace ID                          = 4
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type                = Any data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = No
-- **   Auto-resize enabled                    = No
-- **   Total number of pages                  = 2000
-- **   Number of usable pages                 = 1960
-- **   High water mark (pages)                = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                          1000
, DEVICE '/dev/hdb2'                                          1000
);
-- ******************************************************************************
-- ** start redirect restore
-- ******************************************************************************
RESTORE DATABASE TEST CONTINUE;
-- ******************************************************************************
-- ** end of file
-- ******************************************************************************
```

Usage notes:

- A RESTORE DATABASE command of the form `db2 restore db <name>` will perform a full database restore with a database image and will perform a table space restore operation of the table spaces found in a table space image. A RESTORE DATABASE command of the form `db2 restore db <name> tablespace` performs a table space restore of the table spaces found in the image. In addition, if a list of table spaces is provided with such a command, the explicitly listed table spaces are restored.
- Following the restore operation of an online backup, you must perform a roll-forward recovery.
- If a backup image is compressed, the DB2 database system detects this and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. Otherwise, a check is made to see if a library is stored in the backup image and if it exists it is used. Finally, if there is not library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is to be restored from a backup image (either explicitly by specifying the `COMPRESSION LIBRARY` option or implicitly by performing a normal restore of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was taken on. If the platform the backup was taken on is not the same as the platform that the restore is being done on, the restore operation will fail, even if DB2 normally supports cross-platform restores involving the two systems.
- To restore log files from the backup image that contains them, the LOGTARGET option must be specified, providing the fully qualified and valid path that exists on the DB2 server. If those conditions are satisfied, the restore utility will write the log files from the image to the target path. If a LOGTARGET is specified during a restore of a backup image that does not include logs, the restore oepration will return an error before attempting to restore any table space data. A restore operation will also fail with an error if an invalid, or read-only, LOGTARGET path is specified.

- If any log files exist in the LOGTARGET path at the time the RESTORE DATABASE command is issued, a warning prompt will be returned to the user. This warning will not be returned if WITHOUT PROMPTING is specified.
- During a restore operation where a LOGTARGET is specified, if any log file cannot be extracted, the restore operation will fail and return an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation will fail and an error will be returned. The restore database utility will not overwrite existing log files in the LOGTARGET directory.
- You can also restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path will result in an error. If any problem occurs while restoring log files in this mode of operation, the restore operation will terminate immediately and an error will be returned.
- During an automatic incremental restore operation, only the log files included in the target image of the restore operation will be retrived from the backup image. Any log files included in intermediate images referenced during the incremental restore process will not be extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should only be specified with the final restore command to be issued.
- A backup targeted to be restored to another operating system or another DB2 database version must be an offline backup, and cannot be a delta or an incremental backup image. The same is true for backups to be restored to a later DB2 database version.

Related concepts:

- "Backup and restore operations between different operating systems and hardware platforms" on page 9
- "Developing a backup and recovery strategy" on page 3

Related tasks:

- "Using restore" on page 90

Related reference:

- "CREATE DATABASE command" in *Command Reference*
- "db2move - Database movement tool command" in *Command Reference*

db2Restore - Restore a database or table space

Recreates a damaged or corrupted database that has been backed up using the db2Backup API. 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, or restore table spaces from within a database backup image.

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
```

API and data structure syntax:

```
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;
  char *piComprLibrary;
```

```
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  char *piLogTarget;
  struct db2StoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
} db2RestoreStruct;

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

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

typedef SQL_STRUCTURE db2StoragePathsStruct
{
  char                         **storagePaths;
  db2Uint32 numStoragePaths;
} db2StoragePathsStruct;

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;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  char *piLogTarget;
  db2Uint32 iLogTargetLen;
```

```
  struct db2gStoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
  db2Uint32 iRedirectScriptLen;
} db2gRestoreStruct;

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 db2gStoragePathsStruct
{
  struct db2Char *storagePaths;
  db2Uint32 numStoragePaths;
} db2gStoragePathsStruct;

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

db2Restore API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2RestoreStruct.

pDB2RestoreStruct
Input. A pointer to the db2RestoreStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RestoreStruct data structure parameters:

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

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

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.

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

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 a restore (as a result of a fast reconcile) will be reported.

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. For rebuild cases, this can be an include list or exclude list of table spaces used to rebuild your database. See the `DB2TablespaceStruct` structure. The following restrictions apply:

- The database must be recoverable (for non-rebuild cases only); 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.
- In the case of rebuild, the list must be given for 3 of the 5 rebuild types: `DB2RESTORE_ALL_TBSP_IN_DB_EXC`, `DB2RESTORE_ALL_TBSP_IN_IMG_EXC` and `DB2RESTORE_ALL_TBSP_IN_LIST`

piMediaList
Input. Source media for the backup image. The information provided depends on the value of the locationType parameter. The valid values for locationType parameter (defined in sqlutil header file, located in the include directory) are:

- `SQLU_LOCAL_MEDIA` - Local devices (a combination of tapes, disks, or diskettes).
- `SQLU_XBSA_MEDIA` - XBSA interface. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.
- `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.

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

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.

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 in bytes of the piVendorOptions parameter, which cannot exceed 65535 bytes.

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

iBufferSize
Input. Backup buffer size in 4 KB allocation units (pages). Minimum is 8 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 header file, located in the include directory) 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 a successful return of this call, it is expected that the user will issue another call to this API with the iCallerAction parameter set to the value DB2RESTORE_CONTINUE to continue with the restore.

- 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 header file, located in the include directory) 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_COMPR_LIB - Indicates that the compression library is to be restored. This option cannot be used simultaneously with any other type of restore process. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.
- DB2RESTORE_LOGS - Specifies that only the set of log files contained in the backup image are to be restored. If the backup image does not include log files, the restore operation will fail. If this option is specified, the piLogTarget parameter must also be specified.
- 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 data type 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, the db2Rollforward API must be called before the database can be used.

- DB2RESTORE_GENERATE_SCRIPT - Create a script, that can be used to perform a redirected restore. piRedirectScript must contain a valid file name. The iCallerAction need to be either DB2RESTORE_RESTORE_STORDEF or DB2RESTORE_STORDEF_NOINTERRUPT.

The following values should be used for rebuild operations only:

- DB2RESTORE_ALL_TBSP_IN_DB - Restores the database with all the table spaces known to the database at the time of the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_DB_EXC - Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG - Restores the database with only the table spaces in the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG_EXC - Restores the database with only the table spaces in the image being restored except for those specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_LIST - Restores the database with only the table spaces specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.

NOTE: If the backup image is of a recoverable database then WITHOUT ROLLING FORWARD (DB2RESTORE_NOROLLFWD) cannot be specified with any of the above rebuild actions.

piComprLibrary

Input. Indicates the name of the external library to use to decompress the backup image if the image is compressed. The name must be a fully-qualified path that refers to a file on the server. If the value is a null pointer or a pointer to an empty string, the DB2 database system attempts to use the library stored in the image. If the backup is not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore operation will fail.

piComprOptions

Input. This API parameter describes a block of binary data that will be passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte-reversal or code-page conversion must be handled by the compression library. If the first character of the data block is '@', the remainder of the data is interpreted as the name of a file residing on the server. The DB2 database system then replaces the contents of the piComprOptions and iComprOptionsSize parameters with the contents and size of this file and passes these new values to the initialization routine.

iComprOptionsSize

Input. A four-byte unsigned integer that represents the size of the block of data passed as piComprOptions. The iComprOptionsSize parameter should be zero if and only if the piComprOptions value is a null pointer.

piLogTarget

Input. Specifies the absolute path of a directory on the database server that must be used as the target directory for extracting log files from a backup

image. If this parameter is specified, any log files included in the backup image are extracted into the target directory. If this parameter is not specified, log files included in the backup image are not extracted. To extract only the log files from the backup image, DB2RESTORE_LOGS value should be passed to the iOptions parameter.

piStoragePaths
Input. A structure containing fields that describe a list of storage paths used for automatic storage. Set this to NULL if automatic storage is not enabled for the database.

piRedirectScript
Input. The file name for the redirect restore script that will be created on client side. The file name can be specified relative or absolute. The iOptions field need to have the DB2RESTORE_GENERATE_SCRIPT bit set.

db2TablespaceStruct data structure specific parameters:

tablespaces
Input. 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
Input. Number of entries in the tablespaces parameter.

db2MediaListStruct data structure parameters:

locations
Input. 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
Input. The number of entries in the locations parameter.

locationType
Input. A character indicating the media type. Valid values (defined in sqlutil header file, located in the include directory.) are:

- `SQLU_LOCAL_MEDIA` - Local devices (tapes, disks, diskettes, or named pipes).
- `SQLI_XBSA_MEDIA` - XBSA interface.
- `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).

db2StoragePathsStruct data structure parameters:

storagePaths
Input. An array of strings containing fully qualified names of storage paths on the server that will be used for automatic storage table spaces. In a multi-partition database the same storage paths are used on all database partitions. If a multi-partiton database is being restored with new storage paths, then the catalog partition must be restored before any other database partitions are restored.

numStoragePaths
Input. The number of storage paths in the storagePaths parameter of the db2StoragePathsStruct structure.

db2gRestoreStruct data structure specific parameters:

iSourceDBAliasLen
Input. Specifies the length in bytes of the piSourceDBAlias parameter.

iTargetDBAliasLen
Input. Specifies the length in bytes of the piTargetDBAlias parameter.

iApplicationIdLen
Input. Specifies the length in bytes of the poApplicatoinId parameter. Should be equal to SQLU_APPLID_LEN + 1. The constant SQLU_APPLID_LEN is defined in sqlutil header file that is located in the include directory.

iTimestampLen
Input. Specifies the length in bytes of the piTimestamp parameter.

iTargetDBPathLen
Input. Specifies the length in bytes of the piTargetDBPath parameter.

iReportFileLen
Input. Specifies the length in bytes of the piReportFile parameter.

iUsernameLen
Input. Specifies the length in bytes of the piUsername parameter. Set to zero if no user name is provided.

iPasswordLen
Input. Specifies the length in bytes of the piPassword parameter. Set to zero if no password is provided.

iNewLogPathLen
Input. Specifies the length in bytes of the piNewLogPath parameter.

iLogTargetLen
Input. Specifies the length in bytes of the piLogTarget parameter.

iRedirectScriptLen
Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in piRedirectScript. Set to zero if no script name is given.

db2Char data structure parameters:

pioData
A pointer to a character data buffer. If NULL, no data will be returned.

iLength
Input. The size of the pioData buffer.

oLength
Output. The number of valid characters of data in the pioData buffer.

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. In this case, if the file is replaced, a warning message is returned.
- The database or table space must have been backed up using the db2Backup API.
- If the caller action value is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action value 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 value 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 value 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 value 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 value to DB2RESTORE_PARM_CHK.
- Set caller action value to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with the sqlbstsc API.
- 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 in 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.

- To restore log files from a backup image that contains them, the LOGTARGET option must be specified, assuming a fully qualified and valid path exists on the DB2 server. If those conditions are satisfied, the restore utility writes the log files from the image to the target path. If LOGTARGET is specified during a restoration of a backup image that does not include logs, the restore operation returns an error before attempting to restore any table space data. A restore operation also fails with an error if an invalid or read-only LOGTARGET path is specified.
- If any log files exist in the LOGTARGET path at the time the RESTORE command is issued, a warning prompt is returned to user. This warning is not returned if WITHOUT PROMPTING is specified.
- During a restore operation in which a LOGTARGET is specified, if any log file cannot be extracted, the restore operation fails and returns an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation fails and an error is returned. The restore utility does not overwrite existing log files in the LOGTARGET directory.
- You can restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path results in an error. If any problem occurs while restoring log files in this mode the restore operation terminates immediately and an error is returned.
- During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved from the backup image. Any logs that are included in intermediate images that are referenced during the incremental restore process are not extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should be specified only with the final restore command.
- If a backup is compressed, the DB2 database system detects this state and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. If a library is not specified on the db2Restore API, the library stored in the backup image is used. And if there is no library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is being restored from a backup image (either explicitly by specifying the DB2RESTORE_COMPR_LIB restore type or implicitly by performing a normal restoration of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was taken on. If the platforms are different, the restore operation will fail, even when the DB2 database system normally supports cross-platform restore operations involving the two systems.
- If restoring a database that is enabled for automatic storage, the storage paths associated with the database can be redefined or they can remain as they were previously. To keep the storage path definitions as is, do not provide any storage paths as part of the restore operation. Otherwise, specify a new set of storage paths to associate with the database. Automatic storage table spaces will be automatically redirected to the new storage paths during the restore operation.

Related tasks:

- "Using restore" on page 90

Related reference:

- "sqlemgdb API - Migrate previous version of DB2 database to current version" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "RESTORE DATABASE " on page 100
- "db2Rollforward - Roll forward a database" on page 177
- "db2Backup - Back up a database or table space" on page 76
- "db2Recover - Restore and roll forward a database" on page 199

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 tasks:

- "Using restore" on page 90

Related reference:

- "LIST TABLESPACE CONTAINERS command" in *Command Reference*
- "Rebuild sessions - CLP examples" on page 145
- "SET TABLESPACE CONTAINERS command" in *Command Reference*

Optimizing restore performance

When you perform a restore operation, DB2 will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration. The objective is to minimize the time it takes to complete a restore operation. Unless you explicitly enter a value for the following RESTORE DATABASE command parameters, DB2 will select one for them:

- WITH num-buffers BUFFERS
- PARALLELISM n
- BUFFER buffer-size

For restore operations, a multiple of the buffer size used by the backup operation will always be used. The values specified by the database manager configuration parameters BACKBUFSZ and RESTBUFSZ are ignored. If you want to use these values, you must explicitly specify a buffer size when you issue the RESTORE DATABASE command.

You can also choose to do any of the following 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 8.
- Increase the value of the PARALLELISM parameter.

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

Related concepts:

- "Restore overview" on page 89

Related tasks:

- "Using restore" on page 90

Database rebuild

Rebuilding a database is the process of restoring a database or a subset of its table spaces using a set of restore operations. The functionality provided with database rebuild makes DB2 more robust and versatile, and provides you with a more complete recovery solution.

The ability to rebuild a database from table space backup images means that you no longer have to take as many full database backups. As databases grow in size, opportunities for taking a full database backup are becoming limited. With table space backup as an alternative, you no longer need to take full database backups as frequently. Instead, you can take more frequent table space backups and plan to use them, along with log files, in case of a disaster.

In a recovery situation, if you need to bring a subset of table spaces online faster than others, you can use rebuild to accomplish this. The ability to bring only a subset of table spaces online is especially useful in a test and production environment.

Rebuilding a database involves a series of potentially many restore operations. A rebuild operation can use a database image, or table space images, or both. It can use full backups, or incremental backups, or both. The initial restore operation restores the target image, which defines the structure of the database that can be restored (such as the table space set and the database configuration). For recoverable databases, rebuilding allows you to build a database that is connectable and that contains the subset of table spaces that you need to have online, while keeping table spaces that can be recovered at a later time offline.

The method you use to rebuild your database depends on whether it is recoverable or non-recoverable.

- If the database is recoverable, use one of the following methods:
  - Using a full or incremental database or table space backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image only using the REBUILD option. You can then roll your database forward to a point in time.
  - Using a full or incremental database or table space backup image as your target, rebuild your database by specifying the set of table spaces defined in the database at the time of the target image to be restored using the REBUILD option. SYSCATSPACE must be part of this set. This operation will restore those table spaces specified that are defined in the target image and then use the recovery history file to find and restore any other required backup images for the remaining table spaces not in the target image automatically. Once the restores are complete, roll your database forward to a point in time.
- If the database is non-recoverable:
  - Using a full or incremental database backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image using the appropriate REBUILD syntax. When the restore completes you can connect to the database.

Specifying the target image:

To perform a rebuild of a database, you start by issuing the RESTORE command, specifying the most recent backup image that you use as the target of the restore operation. This image is known as the target image of the rebuild operation,

because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. The rebuild target image is specified using the TAKEN AT parameter in the RESTORE DATABASE command. The target image can be any type of backup (full, table space, incremental, online or offline). The table spaces defined in the database at the time the target image was created will be the table spaces available to rebuild the database.

You must specify the table spaces you want restored using one of the following methods:

- Specify that you want all table spaces defined in the database to be restored and provide an exception list if there are table spaces you want to exclude
- Specify that you want all table spaces that have user data in the target image to be restored and provide an exception list if there are table spaces you want to exclude
- Specify the list of table spaces defined in the database that you want to restore

Once you know the table spaces you want the rebuilt database to contain, issue the RESTORE command with the appropriate REBUILD option and specify the target image to be used.

Rebuild phase:

After you issue the RESTORE command with the appropriate REBUILD option and the target image has been successfully restored, the database is considered to be in the rebuild phase. After the target image is restored, any additional table space restores that occur will restore data into existing table spaces, as defined in the rebuilt database. These table spaces will then be rolled forward with the database at the completion of the rebuild operation.

If you issue the RESTORE command with the appropriate REBUILD option and the database does not exist, a new database is created based on the attributes in the target image. If the database does exist, you will receive a warning message notifying you that the rebuild phase is starting. You will be asked if you want to continue the rebuild operation or not.

The rebuild operation restores all initial metadata from the target image. This includes all data that belongs to the database and does not belong to the table space data or the log files. Examples of initial metadata are:

- Table spaces definitions
- The history file, which is a database file that records administrative operations

The rebuild operation also restores the database configuration. The target image sets the log chain that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

If a database already exists on disk and you want the history file to come from the target image, then you should specify the REPLACE HISTORY FILE option. The history file on disk at this time is used by the automatic logic to find the remaining images needed to rebuild the database.

Once the target image is restored:

- if the database is recoverable, the database is put into roll-forward pending state and all table spaces that you restore are also put into roll-forward pending state. Any table spaces defined in the database but not restored are put in restore pending state.
- If the database is not recoverable, then the database and the table spaces restored will go into normal state. Any table spaces not restored are put in drop pending state, as they can no longer be recovered. For this type of database, the rebuild phase is complete.

For recoverable databases, the rebuild phase ends when the first ROLLFORWARD DATABASE command is issued and the rollforward utility begins processing log records. If a rollforward operation fails after starting to process log records and a restore operation is issued next, the restore is not considered to be part of the rebuild phase. Such restores should be considered as normal table space restores that are not part of the rebuild phase.

Automatic processing:

After the target image is restored, the restore utility determines if there are remaining table spaces that need to be restored. If there are, they are restored using the same connection that was used for running the RESTORE DATABASE command with the REBUILD option. The utility uses the history file on disk to find the most recent backup images taken prior to the target image that contains each of the remaining table spaces that needs to be restored. The restore utility uses the backup image location data stored in the history file to restore each of these images automatically. These subsequent restores, which are table space level restores, can be performed only offline. If the image selected does not belong on the current log chain, an error is returned. Each table space that is restored from that image is placed in roll-forward pending state.

The restore utility tries to restore all required table spaces automatically. In some cases, it will not be able to restore some table spaces due to problems with the history file, or an error will occur restoring one of the required images. In such a case, you can either finish the rebuild manually or correct the problem and re-issue the rebuild.

If automatic rebuilding cannot complete successfully, the restore utility writes to the diagnostics log (db2diag.log) any information it gathered for the remaining restore steps. You can use this information to complete the rebuild manually.

If a database is being rebuilt, only containers belonging to table spaces that are part of the rebuild process will be acquired.

If any containers need to be redefined through redirected restore, you will need to set the new path and size of the new container for the remaining restores and the subsequent rollforward operation.

If the data for a table space restored from one of these remaining images cannot fit into the new container definitions, the table space is put into restore pending state and a warning message is returned at the end of the restore. You can find additional information about the problem in the diagnostic log.

Completing the rebuild phase:

Once all the intended table spaces have been restored you have two options based on the configuration of the database. If the database is non-recoverable, the

database will be connectable and any table spaces restored will be online. Any table spaces that are in drop pending state can no longer be recovered and should be dropped if future backups will be performed on the database.

If the database is recoverable, you can issue the rollforward command to bring the table spaces that were restored online. If SYSCATSPACE has not been restored, the rollforward will fail and this table space will have to be restored before the rollforward operation can begin. This means that during the rebuild phase, SYSCATSPACE must be restored.

Note: In a partitioned database environment, SYSCATSPACE does not exist on non-catalog partitions so it cannot be rebuilt there. However, on the catalog partition, SYSCATSPACE must be one of the table spaces that is rebuilt, or the rollforward operation will not succeed.

Rolling the database forward brings the database out of roll-forward pending state and rolls any table spaces in roll-forward pending state forward. The rollforward utility will not operate on any table space in restore pending state.

The stop time for the rollforward operation must be a time that is later than the end time of the most recent backup image restored during the rebuild phase. An error will occur if any other time is given. If the rollforward operation is not able to reach the backup time of the oldest image that was restored, the rollforward utility will not be able to bring the database up to a consistent point, and the rollforward fails.

You must have all log files for the time frame between the earliest and most recent backup images available for the rollforward utility to use. The logs required are those logs which follow the log chain from the earliest backup image to the target backup image, as defined by the truncation array in the target image, otherwise the rollforward operation will fail. If any backup images more recent than the target image were restored during the rebuild phase, then the additional logs from the target image to the most recent backup image restored are required. If the logs are not made available, the rollforward operation will put those table spaces that were not reached by the logs into restore pending state. You can issue the LIST HISTORY command to show the restore rebuild entry with the log range that will be required by roll forward.

The correct log files must be available. If you rely on the rollforward utility to retrieve the logs, you must ensure that the DB2 Log Manager is configured to indicate the location from which log files can be retrieved. If the log path or archive path has changed, you need to use the OVERFLOW LOG PATH option of the ROLLFORWARD DATABASE command.

Use the AND STOP option of the ROLLFORWARD DATABASE command to make the database available when the rollforward command successfully completes. At this point, the database is no longer in roll-forward pending state. If the rollforward operation begins, but an error occurs before it successfully completes, the rollforward operation stops at the point of the failure and an error is returned. The database remains in roll-forward pending state. You must take steps to correct the problem (for example, fix the log file) and then issue another rollforward operation to continue processing.

If the error cannot be fixed, you will be able to bring the database up at the point of the failure by issuing the ROLLFORWARD STOP command. Any log data beyond that point in the logs will no longer be available once the STOP option is

used. The database comes up at that point and any table spaces that have been recovered are online. Table spaces that have not yet been recovered are in restore pending state. The database is in the normal state.

You will have to decide what is the best way to recover the remaining table spaces in restore pending state. This could be by doing a new restore and roll forward of a table space or by re-issuing the whole rebuild operation again. This will depend on the type of problems encountered. In the situation where SYSCATSPACE is one of the table spaces in restore pending state, the database will not be connectable.

Related concepts:

- "Choosing a target image for database rebuild" on page 134
- "Rebuild and incremental backup images" on page 142

Related tasks:

- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding a partitioned database" on page 140

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Rebuild - details

Choosing a target image for database rebuild

The rebuild target image should be the most recent backup image that you want to use as the starting point of your restore operation. This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. It can be any type of backup (full, table space, incremental, online or offline).

The target image sets the log sequence (or log chain) that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

The following examples illustrate how to choose the image you should use as the target image for a rebuild operation.

Suppose there is a database called SAMPLE that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 15 on page 135 shows that the following database-level backups and table space-level backups that have been taken, in chronological order:

1. Full database backup DB1
2. Full table space backup TS1
3. Full table space backup TS2
4. Full table space backup TS3

5. Database restore and roll forward to a point between TS1 and TS2
6. Full table space backup TS4
7. Full table space backup TS5

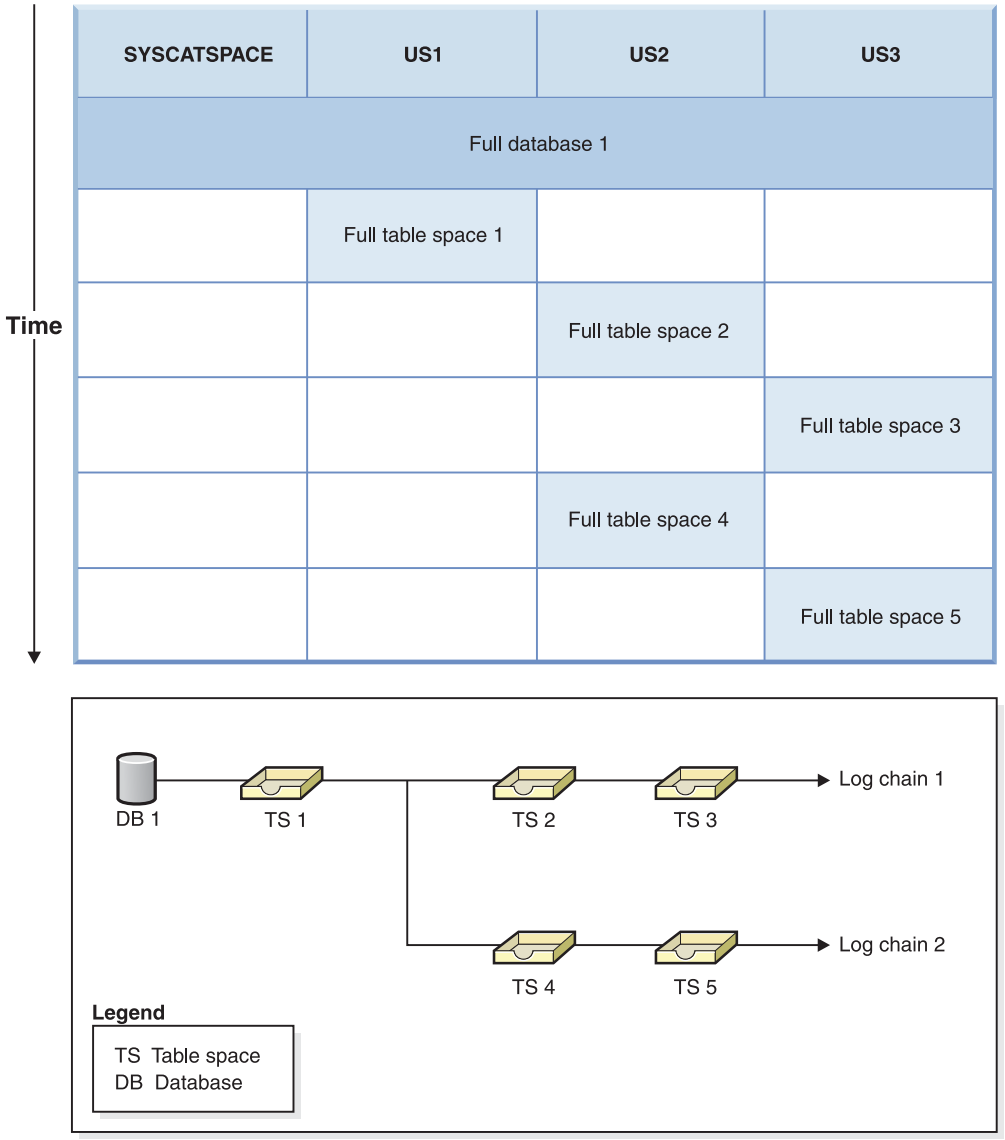


Figure 15. Database and table space-level backups of database SAMPLE

Example 1:

The following example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the current point of time. First you need to choose the table spaces you want to rebuild. Since your goal is to rebuild the database to the current point of time you need to select the most recent backup image as your target image. The most recent backup image is image TS5, which is on log chain 2:

```
db2 restore db sample rebuild with all tablespaces in database taken at
   TS5 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

This restores backup images TS5, TS4, TS1 and DB1 automatically, then rolls the database forward to the end of log chain 2.

Note: All logs belonging to log chain 2 must be accessible for the rollforward operations to complete.

Example 2:

This second example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the end of log chain 1. The target image you select should be the most recent backup image on log chain 1, which is TS3:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at TS3 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

This restores backup images TS3, TS2, TS1 and DB1 automatically, then rolls the database forward to the end of log chain 1.

Note: All logs belonging to log chain 1 must be accessible for the rollforward operations to complete.

Choosing the wrong target image for rebuild:

Suppose there is a database called SAMPLE2 that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Figure 16 shows the backup log chain for SAMPLE2, which consists of the following backups:

1. BK1 is a full database backup, which includes all table spaces
2. BK2 is a full table space backup of USERSP1
3. BK3 is a full table space backup of USERSP2

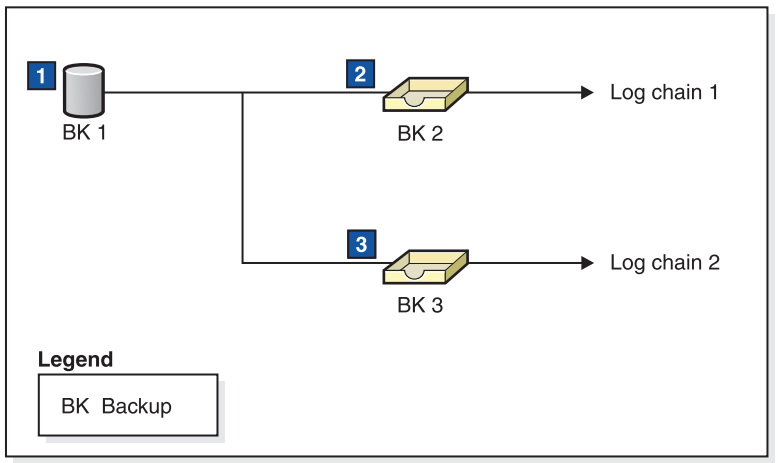


Figure 16. Backup log chain for database SAMPLE2

The following example demonstrates the CLP command you need to issue to rebuild the database from BK3 using table spaces SYSCATSPACE and USERSP2:

```
db2 restore db sample2 rebuild with tablespace (SYSCATSPACE,
   USERSP2) taken at BK3 without prompting
```

Now suppose that after this restore completes, you decide that you also want to restore USERSP1, so, you issue the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK2
```

This restore fails and provides a message that says BK2 is from the wrong log chain (SQL2154N). As you can see in Figure 16 on page 136, the only image that can be used to restore USERSP1 is BK1. Therefore, you need to type the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK1
```

This succeeds so that database can be rolled forward accordingly.

Related concepts:

- "Database rebuild" on page 130

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Restrictions for database rebuild

The following list is a summary of database rebuild restrictions:

- One of the table spaces you rebuild must be SYSCATSPACE on the catalog partition.
- You cannot perform a rebuild operation using the Control Center GUI tools. You must either issue commands using the command line processor (CLP) or use the corresponding application programming interfaces (APIs).
- The REBUILD option cannot be used against a pre-Version 9.1 target image unless the image is that of an offline database backup. If the target image is an offline database backup then only the table spaces in this image can be used for the rebuild. The database will need to be migrated after the rebuild operation successfully completes. Attempts to rebuild using any other type of pre-Version 9.1 target image will result in an error.
- The REBUILD option cannot be issued against a target image from a different operating system than the one being restored on unless the target image is a full database backup. If the target image is a full database backup then only the table spaces in this image can be used for the rebuild. Attempts to rebuild using any other type of target image from a different operating system than the one being restored on will result in an error.

Related concepts:

- "Database rebuild" on page 130

Rebuilding a database using selected table space images

Database rebuild enables you to recreate an entire database using table-space level backup images. Because you can make use of table-space level images to rebuild a database it means that you do not have to rely as heavily on full database backups

Procedure:

To rebuild a database using table-space level backup images, consider the following example.

In this example, there is a recoverable database called SAMPLE with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

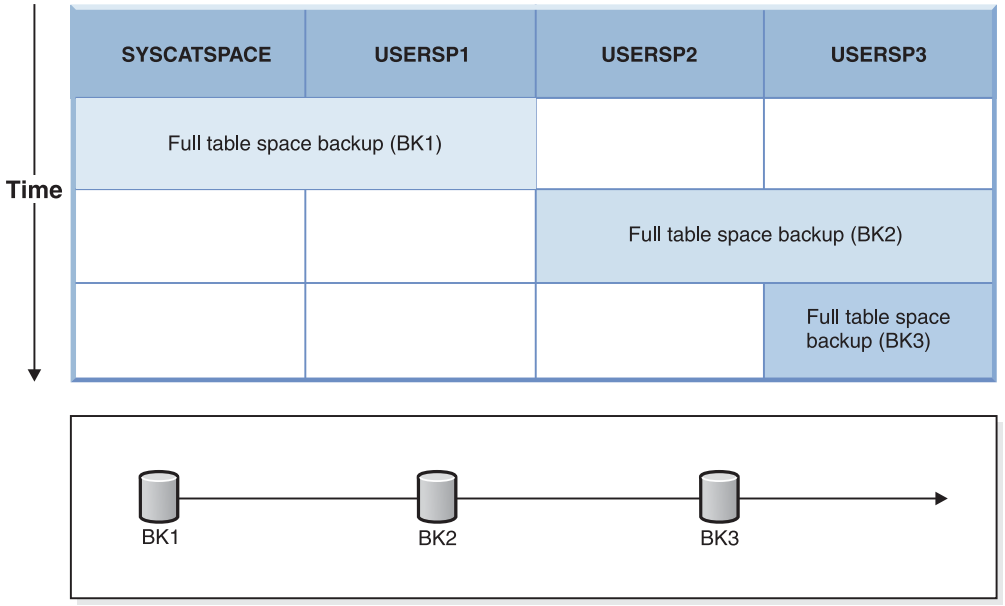


Figure 17. Backup images available for database SAMPLE

The following procedure demonstrates using the RESTORE DATABASE and ROLLFORWARD DATABASE commands, issued through the CLP, to rebuild the database using table space-level backup images:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at BK3 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Related concepts:

- "Rebuild and incremental backup images" on page 142
- "Understanding table space states" on page 60

Related tasks:

- "Rebuilding a partitioned database" on page 140

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Rebuilding selected table spaces

Rebuilding a database allows you to build a database that contains a subset of the table spaces that make up the original database. Rebuilding only a subset of table spaces within a database can be useful in the following situations:

- In a test and development environment in which you want to work on only a subset of table spaces.
- In a recovery situation in which you need to bring table spaces that are more critical online faster than others, you can first restore a subset of table spaces then restore other table spaces at a later time.

Procedure:

To rebuild a database that contains a subset of the table spaces that make up the original database, consider the following example.

In this example, there is a database named SAMPLE that has the following table spaces:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

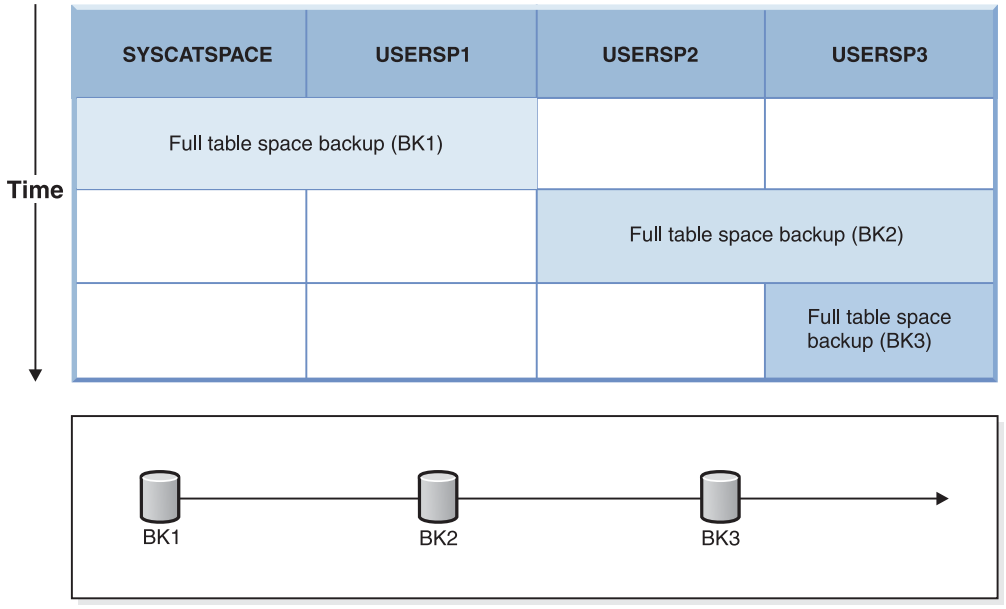


Figure 18. Backup images available for database SAMPLE

The following procedure demonstrates using the RESTORE DATABASE and ROLLFORWARD DATABASE commands, issued through the CLP, to rebuild just SYSCATSPACE and USERSP1 to end of logs:

```
db2 restore db mydb rebuild with all tablespaces in image
   taken at BK1 without prompting
db2 rollforward db mydb to end of logs
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in restore-pending state. You can still restore USERSP2 and USERSP3 at a later time.

Related concepts:

- "Choosing a target image for database rebuild" on page 134
- "Database rebuild" on page 130
- "Rebuild and incremental backup images" on page 142
- "Rebuild and table space containers" on page 143
- "Restrictions for database rebuild" on page 137
- "Understanding table space states" on page 60

Related tasks:

- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding a partitioned database" on page 140

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Rebuilding a partitioned database

Procedure:

To rebuild a partitioned database, rebuild each database partition separately. For each database partition, beginning with the catalog partition, first restore all the table spaces that you require. Any table spaces that are not restored are placed in restore pending state. Once all the database partitions are restored, you then issue the ROLLFORWARD DATABASE command on the catalog partition to roll all of the database partitions forward.

Note: If, at a later date, you need to restore any table spaces that were not originally included in the rebuild phase, you need to make sure that when you subsequently roll the table space forward that the rollforward utility keeps all the data across the database partitions synchronized. If a table space is missed during the original restore and rollforward operation, it might not be detected until there is an attempt to access the data and a data access error occurs. You will then need to restore and roll the missing table space forward to get it back in sync with the rest of the partitions.

To rebuild a partitioned database using table-space level backup images, consider the following example.

In this example, there is a recoverable database called SAMPLE with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BKxy represents backup number x on partition y:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

The following procedure demonstrates using the RESTORE DATABASE and ROLLFORWARD DATABASE commands, issued through the CLP, to rebuild the entire database to the end of logs.

1. On database partition 1, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db sample rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```

2. On database partition 2, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db sample rebuild with tablespaces in database
      taken at BK42 without prompting
   ```

3. On database partition 3, issue a RESTORE DATABASE command with the REBUILD option:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at BK43 without prompting
```

4. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option:
   ```
   db2 rollforward db sample to end of logs
   ```
5. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db sample stop
   ```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Related concepts:

- "Choosing a target image for database rebuild" on page 134
- "Database rebuild" on page 130
- "Rebuild and incremental backup images" on page 142
- "Rebuild and table space containers" on page 143
- "Restrictions for database rebuild" on page 137

Related tasks:

- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding selected table spaces" on page 139

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Rebuild and incremental backup images

You can rebuild a database using incremental images. By default, the restore utility tries to use automatic incremental restore for all incremental images. This means that if you do not use the INCREMENTAL option of the RESTORE DATABASE command, but the target image is an incremental backup image, the restore utility will issue the rebuild operation using automatic incremental restore. If the target image is not an incremental image, but another required image is an incremental image then the restore utility will make sure those incremental images are restored using automatic incremental restore. The restore utility will behave in the same way whether you specify the INCREMENTAL option with the AUTOMATIC option or not.

If you specify the INCREMENTAL option but not the AUTOMATIC option, you will need to perform the entire rebuild process manually. The restore utility will just restore the initial metadata from the target image, as it would in a regular manual incremental restore. You will then need to complete the restore of the target image using the required incremental restore chain. Then you will need to restore the remaining images to rebuild the database.

It is recommended that you use automatic incremental restore to rebuild your database. Only in the event of a restore failure, should you attempt to rebuild a database using manual methods.

Related concepts:

- "Database rebuild" on page 130
- "Choosing a target image for database rebuild" on page 134

- "Rebuild and table space containers" on page 143
- "Restrictions for database rebuild" on page 137

Related tasks:
- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding a partitioned database" on page 140
- "Rebuilding selected table spaces" on page 139

Related reference:
- "Rebuild sessions - CLP examples" on page 145

Rebuild and table space containers

During a rebuild, only those table spaces that are part of the rebuild process will have their containers acquired. The containers belonging to each table space will be acquired at the time the table space user data is restored out of an image.

When the target image is restored, each table space known to the database at the time of the backup will have its definitions only restored. This means the database created by the rebuild will have knowledge of the same table spaces it did at backup time. For those table spaces that should also have their user data restored from the target image, their containers will also be acquired at this time.

Any remaining table spaces that are restored through intermediate table space restores will have their containers acquired at the time the image is restored that contains the table space data.

Rebuild with redirected restore:

In the case of redirected restore, all table space containers must be defined during the restore of the target image. If you specify the REDIRECT option, control will be given back to you to redefine your table space containers. If you have redefined table space containers using the SET TABLESPACE CONTAINERS command then those new containers will be acquired at that time. Any table space containers that you have not redefined will be acquired as normal, at the time the table space user data is restored out of an image.

If the data for a table space that is restored cannot fit into the new container definitions, the table space will be put into restore-pending state and a warning (SQL2563W) will be returned to the you at the end of the restore. There will be a message in the DB2 diagnostics log detailing the problem.

Related concepts:
- "Choosing a target image for database rebuild" on page 134
- "Database rebuild" on page 130
- "Rebuild and incremental backup images" on page 142
- "Restrictions for database rebuild" on page 137

Related tasks:
- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding a partitioned database" on page 140
- "Rebuilding selected table spaces" on page 139

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Rebuild and temporary table spaces

In general, a DB2 backup image is made up of the following components:

- Initial database metadata, such as the table space definitions, database configuration file, and history file.
- Data for non-temporary table spaces specified to the BACKUP utility
- Final database metadata such as the log file header
- Log files (if the INCLUDE LOGS option was specified)

In every backup image, whether it is a database or table space backup, a full or incremental (delta) backup, these core components can always be found.

A database backup image will contain all of the above components, as well as data for every table space defined in the database at the time of the backup.

A table space backup image will always include the database metadata listed above, but it will only contain data for those table spaces that are specified to the backup utility.

Temporary table spaces are treated differently than non-temporary table spaces. Temporary table space data is never backed up, but their existence is important to the framework of the database. Although temporary table space data is never backed up, the temporary table spaces are considered part of the database, so they are specially marked in the metadata that is stored with a backup image. This makes it look like they are in the backup image. In addition, the table space definitions hold information about the existence of any temporary table spaces.

Although no backup image ever contains data for a temporary table space, during a database rebuild operation when the target image is restored (regardless the type of image), temporary table spaces are also restored, only in the sense that their containers are acquired and allocated. The acquisition and allocation of containers is done automatically as part of the rebuild processing. As a result, when rebuilding a database, you cannot exclude temporary table spaces.

Related concepts:

- "Choosing a target image for database rebuild" on page 134
- "Database rebuild" on page 130
- "Rebuild and incremental backup images" on page 142
- "Rebuild and table space containers" on page 143

Related tasks:

- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding a partitioned database" on page 140
- "Rebuilding selected table spaces" on page 139

Related reference:

- "Rebuild sessions - CLP examples" on page 145

Rebuild sessions - CLP examples

Scenario 1:

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

Example 1

The following rebuilds the entire database to the most recent point in time:

1. Issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 2

The following rebuilds just SYSCATSPACE and USERSP2 to a point in time (where end of BK3 is less recent than the point in time, which is less recent than end of logs):

1. Issue a RESTORE DATABASE command with the REBUILD option and specify the table spaces you want to include.

   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      taken at BK2 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO PIT option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to PIT
   ```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

To restore USERSP1 and USERSP3 at a later time, using normal table space restores (without the REBUILD option):

1. Issue the RESTORE DATABASE command *without* the REBUILD option and specify the table space you want to restore. First restore USERSPI:

   ```
   db2 restore db mydb tablespace (USERSP1) taken at BK1 without prompting
   ```

2. Then restore USERSP3:

   ```
   db2 restore db mydb tablespace taken at BK3 without prompting
   ```

3. Issue a ROLLFORWARD DATABASE command with the END OF LOGS option and specify the table spaces to be restored (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs tablespace (USERSP1, USERSP3)
   ```

   The rollforward will replay all logs up to the PIT and then stop for these two table spaces since no work has been done on them since the first rollforward.

4. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

Example 3

The following rebuilds just SYSCATSPACE and USERSP1 to end of logs:

1. Issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with all tablespaces in image
      taken at BK1 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in RESTORE_PENDING state.

Example 4

In the following example, the backups BK1 and BK2 are no longer in the same location as stated in the history file, but this is not known when the rebuild is issued.

1. Issue a RESTORE DATABASE command with the REBUILD option , specifying that you want to rebuild the entire database to the most recent point in time:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 without prompting
   ```

   At this point, the target image is restored successfully, but an error is returned from the restore utility stating it could not find a required image.

2. You must now complete the rebuild manually. Since the database is in the rebuild phase this can be done as follows:

   a. Issue a RESTORE DATABASE command and specify the location of the BK1 backup image:

      ```
      db2 restore db mydb tablespace taken at BK1 from <location>
         without prompting
      ```

   b. Issue a RESTORE DATABASE command and specify the location of the BK2 backup image:

      ```
      db2 restore db mydb tablespace (USERSP2) taken at BK2 from
         <location> without prompting
      ```

c. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

d. Issue a ROLLFORWARD DATABASE command with the STOP option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 5

In this example, table space USERSP3 contains independent data that is needed for generating a specific report, but you do not want the report generation to interfere with the original database. In order to gain access to the data but not affect the original database, you can use REBUILD to generate a new database with just this table space and SYSCATSPACE. SYSCATSPACE is also required so that the database will be connectable after the restore and roll forward operations.

To build a new database with the most recent data in SYSCATSPACE and USERSP3:

1. Issue a RESTORE DATABASE command with the REBUILD option, and specify that table spaces SYSCATSPACE and USERSP3 are to be restored to a new database, NEWDB:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP3)
   taken at BK3 into newdb without prompting
```

2. Issue a ROLLFORWARD DATABASE command on NEWDB with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db newdb to end of logs
```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

```
db2 rollforward db newdb stop
```

At this point the new database is connectable and only SYSCATSPACE and USERSP3 are in NORMAL state. USERSP1 and USERSP2 are in RESTORE_PENDING state.

Note: If container paths are an issue between the current database and the new database (for example, if the containers for the original database need to be altered because the file system does not exist or if the containers are already in use by the original database) then you will need to perform a redirected restore. The example above assumes the default autostorage database paths are used for the table spaces.

Scenario 2:

In the following example, there is a recoverable database called MYDB that has SYSCATSPACE and one thousand user table spaces named Txxxx, where x stands for the table space number (for example, T0001). There is one full database backup image (BK1)

Example 6

The following restores all table spaces except T0999 and T1000:

1. Issue a RESTORE DATABASE command with the REBUILD option:

```
db2 restore db mydb rebuild with all tablespaces in image except
   tablespace (T0999, T1000) taken at BK1 without prompting
```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

```
db2 rollforward db mydb stop
```

At this point the database will be connectable and all table spaces except T0999 and T1000 will be in NORMAL state. T0999 and T1000 will be in RESTORE_PENDING state.

Scenario 3:

The examples in this scenario demonstrate how to rebuild a recoverable database using incremental backups. In the following examples, there is a database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- FULL1 is a full backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA1 is a delta backup of SYSCATSPACE and USERSP1
- INCR1 is an incremental backup of USERSP2 and USERSP3
- DELTA2 is a delta backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA3 is a delta backup of USERSP2
- FULL2 is a full backup of USERSP1

Example 7

The following rebuilds just SYSCATSPACE and USERSP2 to the most recent point in time using incremental automatic restore.

1. Issue a RESTORE DATABASE command with the REBUILD option. The INCREMENTAL AUTO option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
   incremental auto taken at DELTA3 without prompting
```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

Example 8

The following rebuilds the entire database to the most recent point in time using incremental automatic restore.

1. Issue a RESTORE DATABASE command with the REBUILD option. The INCREMENTAL AUTO option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      incremental auto taken at DELTA3 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 9

The following rebuilds the entire database, except for USERSP3, to the most recent point in time.

1. Issue a RESTORE DATABASE command with the REBUILD option. Although the target image is a non-incremental image, the restore utility will detect that the required rebuild chain includes incremental images and it will automatically restore those images incrementally.

   ```
   db2 restore db mydb rebuild with all tablespaces in database except
      tablespace (USERSP3) taken at FULL2 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

Scenario 4:

The examples in this scenario demonstrate how to rebuild a recoverable database using backup images that contain log files. In the following examples, there is a database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Example 10

The following rebuilds the database with just SYSCATSPACE and USERSP2 to the most recent point in time. There is a full online database backup image (BK1), which includes log files.

1. Issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      taken at BK1 logtarget /logs without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs after the end of BK1 have been saved and are accessible):

```
db2 rollforward db mydb to end of logs overflow log path (/logs)
```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 is in RESTORE_PENDING state.

Example 11

The following rebuilds the database to the most recent point in time. There are two full online table space backup images that include log files:

- BK1 is a backup of SYSCATSPACE, using log files 10-45
- BK2 is a backup of USERSP1 and USERSP2, using log files 64-80

1. Issue a RESTORE DATABASE command with the REBUILD option:

```
db2 restore db mydb rebuild with all tablespaces in database
   taken at BK2 logtarget /logs without prompting
```

   The rollforward operation will start at log file 10, which it will always find in the overflow log path if not in the primary log file path. The log range 46-63, since they are not contained in any backup image, will need to be made available for roll forward.

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option, using the overflow log path for log files 64-80:

```
db2 rollforward db mydb to end of logs overflow log path (/logs)
```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 5:

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (0), SMS system catalog (relative container)
- USERSP1 (1) SMS user data table space (relative container)
- USERSP2 (2) DMS user data table space (absolute container /usersp2)
- USERSP3 (3) DMS user data table space (absolute container /usersp3)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

Example 12

The following rebuilds the entire database to the most recent point in time using redirected restore.

1. Issue a RESTORE DATABASE command with the REBUILD option:

```
db2 restore db mydb rebuild with all tablespaces in database
   taken at BK3 redirect without prompting
```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example:

   ```
   db2 set tablespace containers for 3 using (file '/newusersp2' 10000)
   ```

3.
   ```
   db2 set tablespace containers for 4 using (file '/newusersp3' 15000)
   ```

4. Issue a RESTORE DATABASE command with the CONTINUE option:

   ```
   db2 restore db mydb continue
   ```

5. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

6. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 6:

In the following examples, there is a database called MYDB with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BKxy represents backup number x on partition y:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

Example 13

The following rebuilds the entire database to the end of logs.

1. On database partition 1, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```

2. On database partition 2, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with tablespaces in database taken at
      BK42 without prompting
   ```

3. On database partition 3, issue a RESTORE DATABASE command with the REBUILD option:

```
db2 restore db mydb rebuild with all tablespaces in database
   taken at BK43 without prompting
```

4. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (assumes all logs have been saved and are accessible on all database partitions):

   ```
   db2 rollforward db mydb to end of logs
   ```

5. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Example 14

The following rebuilds SYSCATSPACE, USERSP1 and USERSP2 to the most recent point in time.

1. On database partition 1, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```

2. On database partition 2, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with all tablespaces in image taken at
      BK22 without prompting
   ```

3. On database partition 3, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db mydb rebuild with all tablespaces in image taken at
      BK33 without prompting
   ```

   Note: this command omitted USERSP1, which is needed to complete the rebuild operation.

4. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option:

   ```
   db2 rollforward db mydb to end of logs
   ```

5. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db mydb stop
   ```

   The rollforward succeeds and the database is connectable on all database partitions. All table spaces are in NORMAL state, except USERSP3, which is in RESTORE PENDING state on all database partitions on which it exists, and USERSP1, which is in RESTORE PENDING state on database partition 3.

   When an attempt is made to access data in USERSP1 on database partition 3, a data access error will occur. To fix this, USERSP1 will need to be recovered:

   a. On database partitions 3, issue a RESTORE DATABASE command, specifying a backup image that contains USERSP1:

      ```
      db2 restore db mydb tablespace taken at BK23 without prompting
      ```

   b. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option and the AND STOP option:

      ```
      db2 rollforward db mydb to end of logs on dbpartitionnum (3) and stop
      ```

At this point USERSP1 on database partition 3 can have its data accessed since it is in NORMAL state.

Scenario 7:

In the following examples, there is a *nonrecoverable* database called MYDB with the following table spaces:

- SYSCATSPACE (0), SMS system catalog
- USERSP1 (1) SMS user data table space
- USERSP2 (2) DMS user data table space
- USERSP3 (3) DMS user data table space

There is just one backup of the database, BK1:

Example 15

The following demonstrates using rebuild on a nonrecoverable database.

Rebuild the database using only SYSCATSPACE and USERSP1:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1)
   taken at BK1 without prompting
```

Following the restore, the database is connectable. If you issue the LIST TABLESPACES command you see that that SYSCATSPACE and USERSP1 are in NORMAL state, while USERSP2 and USERSP3 are in DELETE_PENDING/ OFFLINE state. You can now work with the two table spaces that are in NORMAL state.

If you want to do a database backup, you will first need to drop USERSP2 and USERSP3 using the DROP TABLESPACE command, otherwise, the backup will fail.

To restore USERSP2 and USERSP3 at a later time, you need to reissue a database restore from BK1.

Related concepts:

- "Choosing a target image for database rebuild" on page 134
- "Database rebuild" on page 130
- "Rebuild and incremental backup images" on page 142
- "Rebuild and table space containers" on page 143
- "Restrictions for database rebuild" on page 137

Related tasks:

- "Rebuilding a database using selected table space images" on page 137
- "Rebuilding a partitioned database" on page 140
- "Rebuilding selected table spaces" on page 139

Chapter 4. Rollforward recovery

This section describes the DB2 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

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 following is one approach you can use to perform rollforward recovery:

1. Invoke the rollforward utility without the STOP option.
2. Invoke 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. Invoke the rollforward utility with the STOP option. After the operation stops, it is not possible to roll additional changes forward.

An alternate approach you can use to perform rollforward recovery is the following:

1. Invoke the rollforward utility with the AND STOP option.
2. The need to take further steps depends on the outcome of the rollforward operation:
   - If it is successful, the rollforward is complete and the database will be connectable and usable. At this point it is not possible to roll additional changes forward.
   - If any errors were returned, take whatever action is required to fix the problem (for example, if there is a missing logfile, find the log file, or if there are retrieve errors, ensure that log archiving is working). Then reissue the rollforward utility with the AND STOP option.

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 can 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 *logarchmeth1* database configuration parameter to a value other than its default of

OFF. When you set *logarchmeth1* to a value other than OFF, the database is placed in backup pending state, and you must take an offline backup of the database before it can be used again.

Note: Entries will be made in the recovery history file for each log file that is used in a rollforward operation.

Related concepts:

- "Understanding recovery logs" on page 33
- "Recovering data with the load copy location file" on page 165

Related reference:

- "Configuration parameters for database logging" on page 37
- "ROLLFORWARD DATABASE " on page 168
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*

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.

Related reference:

- "db2Rollforward - Roll forward a database" on page 177
- "ROLLFORWARD DATABASE " on page 168

Using rollforward

Use the ROLLFORWARD DATABASE command to apply transactions that were recorded in the database log files to a restored database backup image or table space backup image.

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 might 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 invoke only 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 issue a rollforward operation that ends at a specified 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 database partitions 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 might not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all database partitions.)
  - Cancel the in-progress rollforward operation.
- In a partitioned database environment, the rollforward utility must be invoked from the catalog partition of the database.
- Point in time rollforward of a table space is available only from DB2 Version 9 clients. You should migrate any clients running an earlier version of the database product to Version 9 in order to roll a table space forward to a point in time.

Procedure:

The rollforward utility can be invoked through the command line processor (CLP), the Restore wizard 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 Restore wizard:

1. From the Control Center, expand the object tree until you find the database or table space object that you want to restore.
2. Right-click on the object and select Roll-forward from the pop-up menu. The Rollforward wizard opens.

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

Related concepts:

- "Introducing the plug-in architecture for the Control Center" in *Administration Guide: Implementation*
- "Administrative APIs in Embedded SQL or DB2 CLI Programs" in *Administrative API Reference*

Related reference:

- "db2Rollforward - Roll forward a database" on page 177
- "ROLLFORWARD DATABASE " on page 168

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 might 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 process all log files even if they do not contain log records that affect that table space. To skip the log files known not to contain any log records affecting the table space, set the DB2_COLLECT_TS_REC_INFO registry variable to *ON*. This is the default value. To ensure that the information required for skipping log files is collected, the registry variable must be set before the log files are created and used.

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 database partitions. The table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces on all database 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 set integrity 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 set integrity 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 set integrity 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 integrity processing to bring them out of the set integrity 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.)

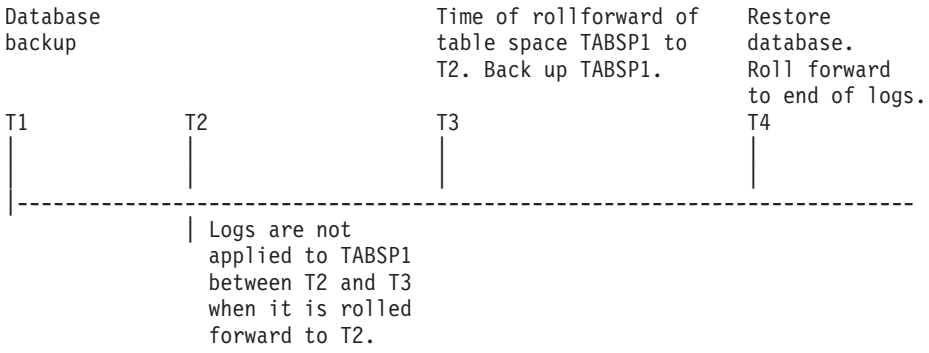


Figure 19. Table Space Backup Requirement

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.

In a database with partitioned tables:

- If you are rolling a table space containing any piece of a partitioned table forward to a point in time, you must also roll all of the other table spaces in which that table resides forward to the same point in time. However, rolling a single table space containing a piece of a partitioned table forward to the end of logs is allowed. If a partitioned table has any attached, detached, or dropped data partitions, then point-in-time rollforward must also include all table spaces for these data partitions. In order to determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSCAT.DATAPARTITIONS catalog view.

Related concepts:

- "Recovering data with the load copy location file" on page 165

Related reference:

- "ROLLFORWARD DATABASE " on page 168
- "SET INTEGRITY statement" in *SQL Reference, Volume 2*

Recovering a dropped table

You might occasionally drop a table that contains 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 can be time-consuming if the database is large, and your data will be unavailable during recovery.

The 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.

The dropped table recovery option is on by default when you create a table space. If you do not want to enable a table space for dropped table recovery, you can either explicitly set the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE command, or you can use the ALTER TABLESPACE command to disable dropped table recovery for an existing table space. The dropped table recovery feature may have a performance impact on forward recovery if there are many drop table operations to recover or if the history file is large.

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.

For partitioned tables, dropped table recovery is always on even if the dropped table recovery is turned off for non-partitioned tables in one or more table spaces. Only one dropped table log record is written for a partitioned table. This log record is sufficient to recover all the data partitions of 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 might contain more information than appeared in the typed table that was dropped.
- XML data. If you attempt to recover a dropped table that contains XML data, the corresponding column data will be empty.

If the table was in reorg pending state when it was dropped, the CREATE TABLE DDL in the history file will not match exactly that of the import file. The import file will be in the format of the table before the first REORG-recommended ALTER was performed, but the **CREATE TABLE** statement in the history file will match the state of the table including the results of any **ALTER TABLE** statements."

If the data being recovered is of the GRAPHIC or VARGRAPHIC data type, it might include more than one code page. In order to recover this data, you need to specify the USEGRAPHICCODEPAGE file type modifier of the IMPORT or LOAD commands. In this case, using the LOAD command to recover the data will increase the performance of the recovery operation.

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 database 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. If the table was in reorg pending state when the drop took place, the contents of the CREATE TABLE DDL might need to be changed to match the contents of the data file.

Related tasks:

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

Related reference:

- "ALTER TABLESPACE statement" in *SQL Reference, Volume 2*
- "CREATE TABLE statement" in *SQL Reference, Volume 2*
- "LIST HISTORY " on page 326
- "ROLLFORWARD DATABASE " on page 168

Recovering data with 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 might be overwritten before rollforward recovery takes place.

Notes:

1. In a multi-partition database, the DB2LOADREC registry variable must be set for all the database partition servers using the **db2set** command.
2. In a multi-partition database, 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
DBPartition    0                      * DB Partition number (OPTIONAL)
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
DBP            18
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            toronto
```

```
BUF             NULL
SES             NULL
TYP             A
TIM             19940325192054
SCH             PAYROLL
TAB             DEPT
DAT             DBT
DB2             toronto
BUF             NULL
SES             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), and DBP (which can be missing from the list).. 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 recreate 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 multi-partition database, the recovery history file is local to each database partition.

Related reference:

• Appendix G, "Tivoli Storage Manager," on page 403

Synchronizing clocks in a partitioned database environment

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 environment, 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, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 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 2005, and the log clock remains at November 7, 2005 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, 2003.

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 partition, the database partition server sends its time to the catalog partition for the database. The catalog partition then checks that the time on the database partition 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:

- "max_time_diff - Maximum time difference among nodes configuration parameter" in *Performance Guide*

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 database partition.
- 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 155
- "Synchronizing clocks in a partitioned database environment" on page 166

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, the *logarchmeth1* or *logarchmeth2* database configuration parameters must be set to a value other than `OFF`) before the database can be recovered with rollforward recovery.

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 database partitions that are listed in the `db2nodes.cfg` file. A database or table space rollforward operation to the end of logs affects the database partitions that are specified. If no database partitions are specified, it affects all database partitions that are listed in the `db2nodes.cfg` file; if rollforward recovery is not needed on a particular partition, that partition is ignored.

For partitioned tables, you are also required to roll forward related table spaces to the same point in time. This applies to table spaces containing data partitions of a table. If a single table space contains a portion of a partitioned table, rolling forward to the end of the logs is still allowed.

Authorization:

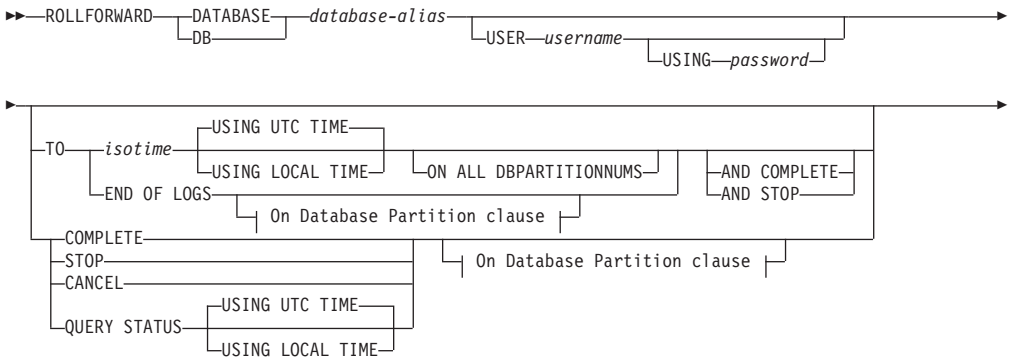
One of the following:

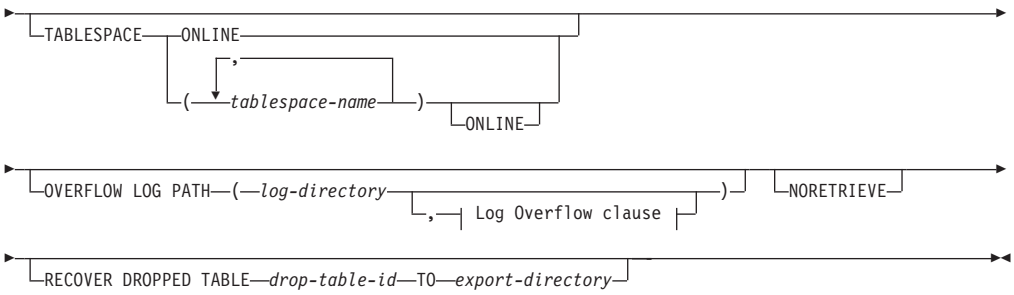
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

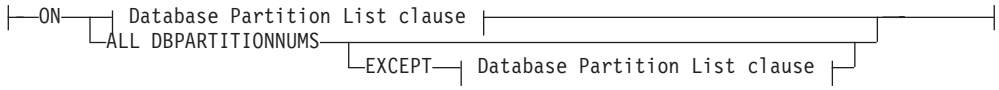
None. This command establishes a database connection.

Command syntax:

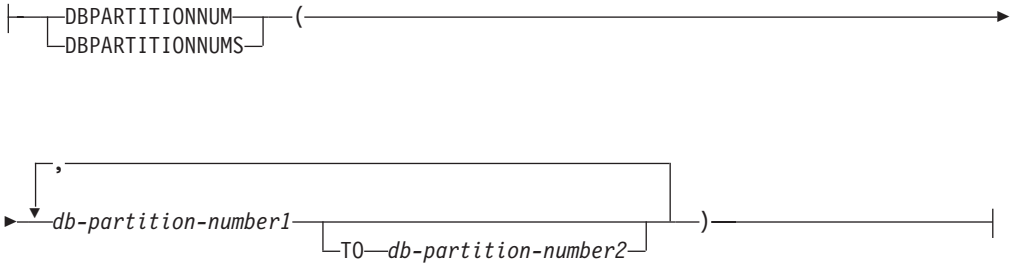




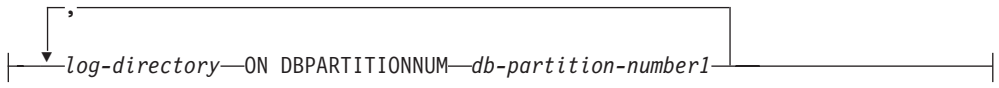
On Database Partition clause:



Database Partition List clause:



Log Overflow clause:



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, you will be 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* (year, month, day, hour, minutes, seconds), expressed in Coordinated Universal Time (UTC, formerly known as GMT). 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 you to rollforward to a point in time that is the server's local time rather than UTC time.

Notes:

1. If you specify a local time for rollforward, all messages returned to you will also be in local time. All times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. The timestamp string is converted to UTC 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.
4. Subsequent ROLLFORWARD commands that cannot specify the USING LOCAL TIME clause will have all messages returned to you in local time if this option is specified.

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 database 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 database 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 database 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`. 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 database 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.

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 UTC) of the last committed transaction since rollforward processing began. In a partitioned database environment, this status information is returned for each database 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*) suffixed by either "UTC" or "Local" (see USING LOCAL TIME). 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.

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 database partition that is rolling forward. If it *does* exist, it must be in the correct state.

For partitioned tables, point in time roll-forward of a table space containing any piece of a partitioned table must also roll-forward all of the other table spaces in which that table resides to the same point in time. Roll-forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached or detached data partitions, then PIT rollforward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it will generally be necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory specified in the ROLLFORWARD DATABASE command. It is possible to

roll forward all table spaces in one command, or do repeated roll forward operations for subsets of the table spaces involved. If the ROLLFORWARD DATABASE command is done for one or a few table spaces, then all data from the table that resided in those table spaces will be recovered. A warning will be written to the notify log if the ROLLFORWARD DATABASE command did not specify the full set of the table spaces necessary to recover all the data for the table. Allowing rollforward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

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 database partitions*. A relative overflow log path can be specified for single-partition databases. 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 database partition.

NORETRIEVE
Allows you to control which log files are to be rolled forward on the standby machine by allowing you to disable the retrieval of archived logs. The benefits of this are:

- By controlling the logfiles to be rolled forward, you can ensure that the standby machine is X hours behind the production machine, to avoid affecting both the 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. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward 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. However if the rollforward AND STOP option is used, and the rollforward encounters an error, the error will be returned to you. In this case, the only way to force the rollforward to stop and come online despite the error (i.e. to come online at that point in the logs before the error) is to issue the rollforward STOP command.

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 and stop
   tablespace(TBS2, TBS3) online
```

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 and stop
   overflow log path (/logs)
```

Example 5 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database 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 (partitioned database environments)

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 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. 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 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (partitioned database environment)

After restoring a table space on all database partitions, roll forward to point in time 2, but do not specify AND STOP. The rollforward operation is still in progress. Cancel and roll forward to point in time 1:

```
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 (partitioned database environments)

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 might require a load recovery using tape devices. If prompted for another tape, you 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. Take all affected tablespaces offline, but continue rollforward processing.

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.
- Point in time rollforward is not supported with pre-V9.1 clients due to V9.1 support for partitioned tables.

Related concepts:

- "Developing a backup and recovery strategy" on page 3

Related tasks:

- "Using rollforward" on page 157

db2Rollforward - Roll forward a 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 the logarchmeth1 database configuration parameter or the logarchmeth2 database configuration parameter must be set to a value other than OFF) before the database can be recovered with rollforward 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.

For partitioned tables, you are also required to roll forward related table spaces to the same point in time. Related table spaces contain attached, detached, and dropped data partitions or indexes of a table. Rollforward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

None. This API establishes a database connection.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Rollforward (
       db2Uint32 versionNumber,
       void * pDB2RollforwardStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
```

```
  struct db2RfwdInputStruct *piRfwdInput;
  struct db2RfwdOutputStruct *poRfwdOutput;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
  sqluint32 iVersion;
  char *piDbAlias;
  db2Uint32 iCallerAction;
  char *piStopTime;
  char *piUserName;
  char *piPassword;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2Uint32 iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piDroppedTblID;
  char *piExportDir;
  db2Uint32 iRollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char *poApplicationId;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  db2Uint32 oRollforwardFlags;
} db2RfwdOutputStruct;

SQL_STRUCTURE sqlurf_newlogpath
{
   SQL_PDB_NODE_TYPE nodenum;
   unsigned short  pathlen;
   char logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
   sqlint32 num_entry;
   struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;

typedef SQL_STRUCTURE sqlu_tablespace_entry
{
   sqluint32 reserve_len;
   char tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
   char filler[1];
} sqlu_tablespace_entry;

SQL_STRUCTURE sqlurf_info
{
   SQL_PDB_NODE_TYPE nodenum;
   sqlint32 state;
   unsigned char   nextarclog[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   firstarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastcommit[SQLUM_TIMESTAMP_LEN+1];
};

SQL_API_RC SQL_API_FN
  db2gRollforward (
       db2Uint32 versionNumber,
```

```
       void * pDB2gRollforwardStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct *piRfwdInput;
  struct db2RfwdOutputStruct *poRfwdOutput;
} db2gRollforwardStruct;

typedef SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32 iDbAliasLen;
  db2Uint32 iStopTimeLen;
  db2Uint32 iUserNameLen;
  db2Uint32 iPasswordLen;
  db2Uint32 iOvrflwLogPathLen;
  db2Uint32 iDroppedTblIDLen;
  db2Uint32 iExportDirLen;
  sqluint32 iVersion;
  char *piDbAlias;
  db2Uint32 iCallerAction;
  char *piStopTime;
  char *piUserName;
  char *piPassword;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2Uint32 iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piDroppedTblID;
  char *piExportDir;
  db2Uint32 iRollforwardFlags;
} db2gRfwdInputStruct;
```

db2Rollforward 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.

db2RollforwardStruct data structure parameters:

piRfwdInput
Input. A pointer to the db2RfwdInputStruct structure.

poRfwdOutput
Output. A pointer to the db2RfwdOutputStruct structure.

db2RfwdInputStruct data structure parameters:

iVersion
Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

piDbAlias
Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_ROLLFWD
Rollforward to the point in time specified by the piStopTime parameter. 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.

DB2ROLLFORWARD_STOP
End roll-forward recovery by rolling forward the database using available log files and then rolling it back. Uncommitted transactions are backed out and the rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_RFWD_STOP
Rollforward to the point in time specified by piStopTime, and end roll-forward recovery. The rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_QUERY
Query values for nextarclog, firstarcdel, lastarcdel, and lastcommit. Return database status and a node number.

DB2ROLLFORWARD_PARM_CHECK
Validate parameters without performing the roll forward.

DB2ROLLFORWARD_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:

DB2ROLLFORWARD_LOADREC_CONT
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2ROLLFORWARD_DEVICE_TERM
Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2ROLLFORWARD_LOAD_REC_TERM
Terminate all devices being used by load recovery.

piStopTime
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 DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_xxx) caller actions.

piUserName
Input. A string containing the user name of the application. Can be NULL.

piPassword
Input. A string containing the password of the supplied user name (if any). Can be NULL.

piOverflowLogPath
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 database 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.

iNumChngLgOvrflw
Input. 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.

piChngLogOvrflw
Input. 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.

iConnectMode
Input. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_OFFLINE
Offline roll forward. This value must be specified for database roll-forward recovery.

DB2ROLLFORWARD_ONLINE
Online roll forward.

piTablespaceList
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.

For partitioned tables, point in time (PIT) roll-forward of a table space containing any piece of a partitioned table must also roll forward all of the other table spaces in which that table resides to the same point in time. Roll forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached, detached or dropped data partitions, then PIT roll-forward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it is generally necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory specified in the piExportDir parameter. It is possible to roll forward all table spaces in one command, or do repeated roll-forward operations for subsets of the table spaces involved. A warning will be written to the notify log if the db2Rollforward API did not specify the full set of the table spaces necessary to recover all the data for the table. A warning will be returned to the user with full details of all partitions not recovered on the command found in the administration notification log.

Allowing the roll forward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES
Apply to all database partition servers. This is the default value. The piNodeList parameter must be set to NULL, if this value is used.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY
Apply to the catalog partition only. The piNodeList parameter must be set to NULL, if this value is used.

iNumNodes
Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the roll-forward recovery.

iNumNodeInfo
Input. Defines the size of the output parameter poNodeInfo, 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 the same as the number of database partition servers for which this API is being called.

piDroppedTblID
Input. A string containing the ID of the dropped table whose recovery is

being attempted. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward command.

piExportDir
Input. The name of the directory into which the dropped table data will be exported.

iRollforwardFlags
Input. Specifies the rollforward flags. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_EMPTY_FLAG
No flags specified.

DB2ROLLFORWARD_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.

DB2ROLLFORWARD_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.

db2RfwdOutputStruct data structure parameters:

poApplicationId
Output. The application ID.

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

oRollforwardFlags
Output. Rollforward output flags. Valid values are:

DB2ROLLFORWARD_OUT_LOCAL_TIME
Indicates to user that the last committed transaction timestamp is displayed in local time rather than UTC. Local time is based on the server's local time, not on the client's. In a partitioned database environment, local time is based on the catalog partition's local time.

sqlurf_newlogpath data structure parameters:

nodenum
Input. The number of the database partition that this structure details.

pathlen
Input. The total length of the logpath field.

logpath
Input. A fully qualified path to be used for a specific node for the rollforward operation.

sqlu_tablespace_bkrst_list data structure parameters:

num_entry
Input. The number of structures contained in the list pointed to by the tablespace parameter.

tablespace
Input. A pointer to a list of sqlu_tablespace_entry structures.

sqlu_tablespace_entry data structure parameters:

reserve_len
Input. Specifies the length in bytes of the tablespace_entry parameter.

tablespace_entry
Input. The name of the table space to rollforward.

filler Filler used for proper alignment of data structure in memory.

sqlurf_info data structure parameters:

nodenum
Output. The number of the database partition that this structure contains information for.

state Output. The current state of the database or tablespaces that were included in the rollfoward on a database partition.

nextarclog
Output. If the rollforward has completed, this field will be empty. If the rollforward has not yet completed, this will be the name of the next log file which will be processed for the rollforward.

firstarcdel
Output. The first log file replayed by the rollforward.

lastarcdel
Output. The last log file replayed by the rollforward.

lastcommit
Output. The time of the last committed transaction.

db2gRfwdInputStruct data structure specific parameters:

iDbAliasLen
Input. Specifies the length in bytes of the database alias.

iStopTimeLen
Input. Specifies the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

iUserNameLen
Input. Specifies the length in bytes of the user name. Set to zero if no user name is provided.

iPasswordLen
Input. Specifies the length in bytes of the password. Set to zero if no password is provided.

iOverflowLogPathLen
Input. Specifies the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

iDroppedTblIDLen
Input. Specifies the length in bytes of the dropped table ID (piDroppedTblID parameter). Set to zero if no dropped table ID is provided.

iExportDirLen
Input. Specifies the length in bytes of the dropped table export directory (piExportDir parameter). Set to zero if no dropped table export directory is provided.

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 DB2ROLLFORWARD_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 lascommit 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 piStopTime, 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 DB2ROLLFORWARD_STOP or DB2ROLLFORWARD_RFWRD_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 DB2ROLLFORWARD_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 tasks:

- "Using rollforward" on page 157

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "ROLLFORWARD DATABASE " on page 168

- "db2Backup - Back up a database or table space" on page 76
- "db2Restore - Restore a database or table space" on page 115
- "db2Recover - Restore and roll forward a database" on page 199

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 are:

```
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 you stop it, so that you do not miss any logs.

If the rollforward command encounters an error, the rollforward operation will not complete. The error will be returned, and you will then be able to fix the error and reissue the command. If, however, you are unable to fix the error, you can force the rollforward to complete by issuing the following:

```
db2 rollforward db sample complete
```

This command brings the database online at the point in the logs before the failure.

Example 2

Roll the database 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 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database 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 offline on database partitions 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 (partitioned database environments)

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 dbpartitionnum 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 environments)

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 dbpartitionnums **

db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

Example 8 (partitioned database environments)

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 environments)

Rollforward recover six small table spaces that reside on a single database 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.

Example 10 (Partitioned tables - Rollforward to end of log on all data partitions)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. All table spaces can be rolled forward to END OF LOGS.

```
db2 rollforward db PBARDB to END OF LOGS and stop
   tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 11 (Partitioned tables - Rollforward to end of logs on one table space)

A partitioned table is created initially using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. Table space tbsp4 becomes corrupt and requires a restore and rollforward to end of logs.

```
db2 rollforward db PBARDB to END OF LOGS and stop tablespace(tbsp4)
```

This completes successfully.

Example 12 (Partitioned tables - Rollforward to PIT of all data partitions including those added, attached, detached or with indexes)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, attaches data partitions from the table in tbsp5, and detaches data partitions from tbsp1. The user performs a rollforward to PIT with all the table spaces used by the partitioned table including those table spaces specified in the INDEX IN clause.

```
db2 rollforward db PBARDB to 2005-08-05-05.58.53.000000 and stop
   tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 13 (Partitioned tables - Rollforward to PIT on a subset of the table spaces)

A partitioned table is created using three table spaces (tbsp1, tbsp2, tbsp3). Later, the user detaches all data partitions from tbsp3. The rollfoward to PIT is only permitted on tbsp1 and tbsp2.

```
db2 rollforward db PBARDB to 2005-08-05-06.02.42.000000 and stop
   tablespace( tbsp1, tbsp2)
```

This completes successfully.

Related concepts:

- "Rollforward overview" on page 155

Related reference:

- "db2Rollforward - Roll forward a database" on page 177
- "ROLLFORWARD DATABASE " on page 168

Chapter 5. Database recover

This section describes the DB2 recover utility, which performs the neccessary restore and rollforward operations to recover a database to a specified time, based on information found in the recovery history file.

The following topics are covered:

Recover overview

The recover utility performs the necessary restore and rollforward operations to recover a database to a specified time, based on information found in the recovery history file. When you use this utility, you specify that the database be recovered to a point-in-time or to the end of the log files. The utility will then select the best suitable backup image and perform the recovery operations.

The recover utility does not support the following RESTORE DATABASE command options:

- TABLESPACE tablespace-name. Table space restore operations are not supported.
- INCREMENTAL. Incremental restore operations are not supported.
- OPEN num-sessions SESSIONS. You cannot indicate the number of I/O sessions that are to be used with TSM or another vendor product.
- BUFFER buffer-size. You cannot set the size of the buffer used for the restore operation.
- DLREPORT filename. You cannot specify a file name for reporting files that become unlinked.
- WITHOUT ROLLING FORWARD. You cannot specify that the database is not to be placed in rollforward pending state after a successful restore operation.
- PARALLELISM n. You cannot indicate the degree of parallelism for the restore operation.
- WITHOUT PROMPTING. You cannot specify that a restore operation is to run unattended

In addition, the recover utility does not allow you to specify any of the REBUILD options. However, the recovery utility will automatically use the appropriate REBUILD option if it cannot locate any database backup images based on the information in the recovery history file.

Related concepts:

- "Client/server timestamp conversion" on page 167

Related tasks:

- "Using recover" on page 192

Related reference:
- "Configuration parameters for database logging" on page 37

Privileges, authorities, and authorization required to use recover

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 recover utility.

Related reference:
- "db2Recover - Restore and roll forward a database" on page 199
- "RECOVER DATABASE" on page 193

Using recover

Use the RECOVER DATABASE command to recover a database to a specified time, using information found in the recovery history file.

If you issue the RECOVER DATABASE command following an incomplete recover operation that ended during the rollforward phase, the recover utility will attempt to continue the previous recover operation, without redoing the restore phase. If you want to force the recover utility to redo the restore phase, issue the RECOVER DATABASE command with the RESTART option to force the recover utility to ignore any prior recover operation that failed to complete. If you are using the application programming interface (API), specify the caller action DB2RECOVER_RESTART for the iRecoverAction field to force the recover utility to redo the restore phase.

If the RECOVER DATABASE command is interrupted during the restore phase, it cannot be continued. You need to reissue the RECOVER DATABASE command.

Prerequisites:

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

The database can be local or remote.

Procedure:

You can invoke the recover utility through the command line processor (CLP) or the **db2Recover** application programming interface (API).

The following example shows how to use the RECOVER DATABASE command through the CLP:

```
db2 recover db sample
```

Note: In a partitioned database environment, the recover utility must be invoked from the catalog partition of the database.

Related concepts:

- "Recover overview" on page 191

Related reference:

- "RECOVER DATABASE" on page 193
- "db2Recover - Restore and roll forward a database" on page 199

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 155
- "Synchronizing clocks in a partitioned database environment" on page 166

RECOVER DATABASE

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope:

In a partitioned database environment, this command can only be invoked from the catalog partition. A database recover operation to a specified point in time affects all database partitions that are listed in the `db2nodes.cfg` file. A database recover operation to the end of logs affects the database partitions that are specified. If no partitions are specified, it affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization:

To recover an existing database, one of the following:

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

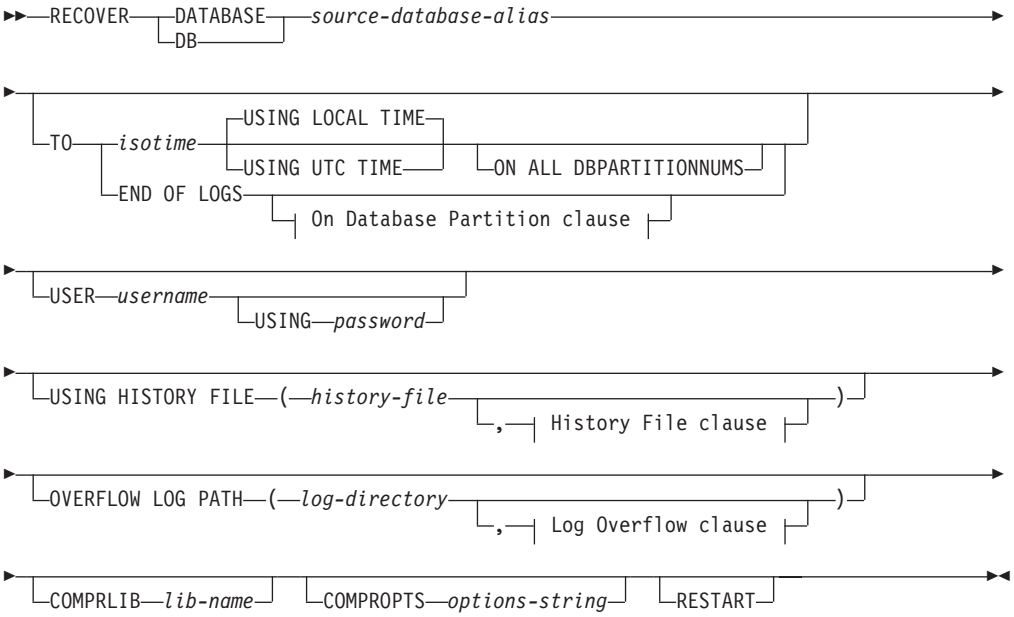
To recover to a new database, one of the following:

- *sysadm*
- *sysctrl*

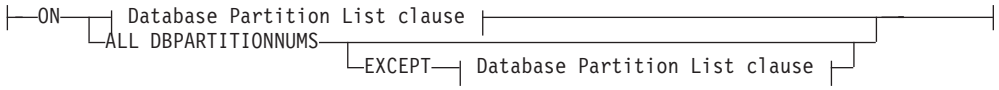
Required connection:

To recover an existing database, a database connection is required. This command automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. To recover to a new database, an instance attachment and a database connection are required. The instance attachment is required to create the database.

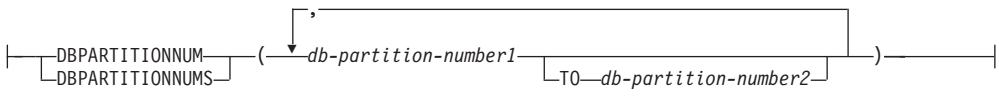
Command syntax:



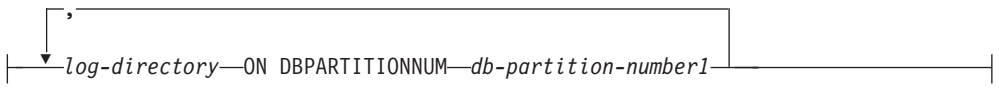
On Database Partition clause:



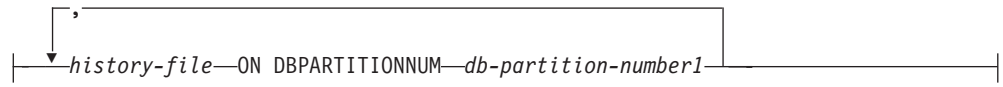
Database Partition List clause:



Log Overflow clause:



History File clause:



Command parameters:

DATABASE *database-alias*
The alias of the database that is to be recovered.

USER *username*
The user name under which the database is to be 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 recovered (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, formerly known as GMT). 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
Specifies the point in time to which to recover. This option allows the user to recover to a point in time that is the server's local time rather than UTC time.

Notes:

1. If the user specifies a local time for recovery, all messages returned to the user will also be in local time. All times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. The timestamp string is converted to UTC 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 saving time, it is important to know if the stop time is before or after the clock change, and specify it correctly.

USING UTC TIME
Specifies the point in time to which to recover.

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.

ON ALL DBPARTITIONNUMS
Specifies that transactions are to be rolled forward on all database 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 database 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 database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

USING HISTORY FILE *history-file*

history-file **ON DBPARTITIONNUM**
In a partitioned database environment, allows a different history file

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 database partitions*. A relative overflow log path can be specified for single-partition databases.

The OVERFLOW LOG PATH command parameter will overwrite the value (if any) of the database configuration parameter *overflowlogpath*.

COMPRLIB *lib-name*
Indicates the name of the library to be used to perform the decompression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *options-string*
Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is "@", the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the

contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

RESTART
The RESTART keyword can be used if a prior RECOVER operation was interrupted or otherwise did not complete. Starting in v91, a subsequent **RECOVER command** will attempt to continue the previous **RECOVER**, if possible. Using the RESTART keyword forces **RECOVER** to start with a fresh restore and then rollforward to the PIT specified.

log-directory **ON DBPARTITIONNUM**
In a partitioned database environment, allows a different log path to override the default overflow log path for a specific database partition.

Examples:

In a single-partition database environment, where the database being recovered currently exists, and so the most recent version of the history file is available in the *dftdbpath*:

1. To use the latest backup image and rollforward to the end of logs using all default values:
```
RECOVER DB SAMPLE
```
2. To recover the database to a PIT, issue the following. The most recent image that can be used will be restored, and logs applied until the PIT is reached.
```
RECOVER DB SAMPLE TO  2001-12-31-04.00.00
```
3. To recover the database using a saved version of the history file. issue the following. For example, if the user needs to recover to an extremely old PIT which is no longer contained in the current history file, the user will have to provide a version of the history file from this time period. If the user has saved a history file from this time period, this version can be used to drive the recover.
```
RECOVER DB SAMPLE TO  1999-12-31-04.00.00
   USING HISTORY FILE (/home/user/old1999files/db2rhist.asc)
```

In a single-partition database environment, where the database being recovered does not exist, you must use the USING HISTORY FILE clause to point to a history file.

1. If you have not made any backups of the history file, so that the only version available is the copy in the backup image, the recommendation is to issue a RESTORE followed by a ROLLFORWARD. However, to use RECOVER, you would first have to extract the history file from the image to some location, for example /home/user/oldfiles/db2rhist.asc, and then issue this command. (This version of the history file does not contain any information about log files that are required for rollforward, so this history file is not useful for RECOVER.)
```
RECOVER DB SAMPLE TO END OF LOGS
   USING HISTORY FILE (/home/user/fromimage/db2rhist.asc)
```
2. If you have been making periodic or frequent backup copies of the history, the USING HISTORY clause should be used to point to this version of the history file. If the file is /home/user/myfiles/db2rhist.asc, issue the command:
```
RECOVER DB SAMPLE TO PIT
   USING HISTORY FILE (/home/user/myfiles/db2rhist.asc)
```

   (In this case, you can use any copy of the history file, not necessarily the latest, as long as it contains a backup taken before the point-in-time (PIT) requested.)

In a partitioned database envrionment, where the database exists on all database partitions, and the latest history file is available on *dftdbpath* on all database partitions:

1. To recover the database to a PIT on all nodes. DB2 will verify that the PIT is reachable on all nodes before starting any restore operations.
   ```
   RECOVER DB SAMPLE TO  2001-12-31-04.00.00
   ```
2. To recover the database to this PIT on all nodes. DB2 will verify that the PIT is reachable on all nodes before starting any restore operations. The RECOVER operation on each node is identical to a single-partition RECOVER.
   ```
   RECOVER DB SAMPLE TO END OF LOGS
   ```
3. Even though the most recent version of the history file is in the *dftdbpath*, you might want to use several specific history files. Unless otherwise specified, each database partition will use the history file found locally at /home/user/oldfiles/db2rhist.asc. The exceptions are nodes 2 and 4. Node 2 will use: /home/user/node2files/db2rhist.asc, and node 4 will use: /home/user/node4files/db2rhist.asc.
   ```
   RECOVER DB SAMPLE TO  1999-12-31-04.00.00
      USING HISTORY FILE (/home/user/oldfiles/db2rhist.asc,
         /home/user/node2files/db2rhist.asc ON DBPARTITIONNUM 2,
         /home/user/node4files/db2rhist.asc ON DBPARTITIONNUM 4)
   ```
4. It is possible to recover a subset of nodes instead of all nodes, however a PIT RECOVER can not be done in this case, the recover must be done to EOL.
   ```
   RECOVER DB SAMPLE TO END OF LOGS ON DBPARTITIONNUMS(2 TO 4, 7, 9)
   ```

In a partitioned database environment, where the database does not exist:

1. If you have not made any backups of the history file, so that the only version available is the copy in the backup image, the recommendation is to issue a RESTORE followed by a ROLLFORWARD. However, to use RECOVER, you would first have to extract the history file from the image to some location, for example, /home/user/oldfiles/db2rhist.asc, and then issue this command. (This version of the history file does not contain any information about log files that are required for rollforward, so this history file is not useful for the recover.)
   ```
   RECOVER DB SAMPLE TO PIT
      USING HISTORY FILE (/home/user/fromimage/db2rhist.asc)
   ```
2. If you have been making periodic or frequent backup copies of the history, the USING HISTORY clause should be used to point to this version of the history file. If the file is /home/user/myfiles/db2rhist.asc, you can issue the following command:
   ```
   RECOVER DB SAMPLE TO END OF LOGS
      USING HISTORY FILE (/home/user/myfiles/db2rhist.asc)
   ```

Usage notes:

- Recovering a database might 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 there is a failure during the restore portion of the recover operation, you can reissue the RECOVER DATABASE command. If the restore operation was

successful, but there was an error during the rollforward operation, you can issue a ROLLFORWARD DATABASE command, since it is not necessary (and it is time-consuming) to redo the entire recover operation.
- In a partitioned database environment, if there is an error during the restore portion of the recover operation, it is possible that it is only an error on a single database partition. Instead of reissuing the RECOVER DATABASE command, which restores the database on all database partitions, it is more efficient to issue a RESTORE DATABASE command for the database partition that failed, followed by a ROLLFORWARD DATABASE command.

Related concepts:
- "Developing a backup and recovery strategy" on page 3

Related tasks:
- "Using recover" on page 192

db2Recover - Restore and roll forward a database

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. If no database partition servers are specified, it affects all database partition servers that are listed in the db2nodes.cfg file. If a point in time is specified, the API affects all database partitions.

Authorization:

To recover an existing database, one of the following:
- sysadm
- sysctrl
- sysmaint

To recover to a new database, one of the following:
- sysadm
- sysctrl

Required connection:

To recover an existing database, a database connection is required. This API automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. Instance and database, to recover to a new database. The instance attachment is required to create the database.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Recover (
       db2Uint32 versionNumber,
```

```
       void * pDB2RecovStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RecoverStruct
{
  char *piSourceDBAlias;
  char *piUsername;
  char *piPassword;
  db2Uint32 iRecoverCallerAction;
  db2Uint32 iOptions;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  char *piStopTime;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piHistoryFile;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2RecoverStruct;

SQL_STRUCTURE sqlu_histFile
{
   SQL_PDB_NODE_TYPE nodeNum;
   unsigned short  filenameLen;
   char filename[SQL_FILENAME_SZ+1];
};

SQL_API_RC SQL_API_FN
  db2gRecover (
       db2Uint32 versionNumber,
       void * pDB2gRecoverStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRecoverStruct
{
  char *piSourceDBAlias;
  db2Uint32 iSourceDBAliasLen;
  char *piUserName;
  db2Uint32 iUserNameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  db2Uint32 iRecoverCallerAction;
  db2Uint32 iOptions;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  char *piStopTime;
  db2Uint32 iStopTimeLen;
  char *piOverflowLogPath;
  db2Uint32 iOverflowLogPathLen;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piHistoryFile;
  db2Uint32 iHistoryFileLen;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
```

```
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2gRecoverStruct;
```

db2Recover API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2RecoverStruct.

pDB2RecoverStruct
Input. A pointer to the db2RecoverStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RecoverStruct data structure parameters:

piSourceDBAlias
Input. A string containing the database alias of the database to be recovered.

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

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

iRecoverCallerAction
Input. Valid values are:

DB2RECOVER
Starts the recover operation. Specifies that the recover will run unattended, and that scenarios that 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 recover have been mounted, and utility prompts are not desired.

DB2RECOVER_RESTART
Allows the user to ignore a prior recover and start over from the beginning.

DB2RECOVER_CONTINUE
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2RECOVER_LOADREC_TERM
Terminate all devices being used by load recovery.

DB2RECOVER_DEVICE_TERM
Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2RECOVER_PARM_CHK_ONLY
Used to validate parameters without performing a recover operation. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

DB2RECOVER_DEVICE_TERMINATE
Removes a particular device from the list of devices used by the recover operation. When a particular device has exhausted its input, recover will return a warning to the caller. Call the recover utility again with this caller action to remove the device that generated the warning from the list of devices being used.

iOptions
Input. Valid values are:

- DB2RECOVER_EMPTY_FLAG
No flags specified.

- DB2RECOVER_LOCAL_TIME
Indicates that the value specified for the stop time by piStopTime is in local time, not GMT. This is the default setting.

- DB2RECOVER_GMT_TIME
This flag indicates that the value specified for the stop time by piStopTime is in GMT (Greenwich Mean Time).

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

piStopTime
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 DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_) caller actions.

piOverflowLogPath
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 location specified by the logpath configuration parameter before they can be used by this utility. This can be a problem if the user does not have sufficient space in the log path. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the log path, and then in the overflow log path. The log files needed for table space rollforward recovery can be brought into either the log path or the overflow log path. If the caller does not specify an overflow log path, the default value is the log path.

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 database partition. In a single-partition database environment, the overflow log path can be relative if the server is local.

iNumChngLgOvrflw
Input. 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.

piChngLogOvrflw
Input. 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.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES
Apply to all database partition servers. piNodeList should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY
Apply to the catalog partition only. piNodeList should be NULL.

iNumNodes
Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the rollforward recovery.

iNumNodeInfo
Input. Defines the size of the output parameter poNodeInfo, 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 the same as the number of database partition servers for which this API is being called.

piHistoryFile
History file.

iNumChngHistoryFile
Number of history files in list.

piChngHistoryFile
List of history files.

piComprLibrary
Input. Indicates the name of the external library to be used to perform decompression of the backup image if the image is compressed. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore will fail.

piComprOptions
Input. Describes a block of binary data that will be passed to the initialization routine in the decompression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or

code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize
Input. Represents the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

sqlu_histFile data structure parameters:

nodeNum
Input. Specifies which database partition this entry should be used for.

filenameLen
Input. Length in bytes of filename.

filename
Input. Path to the history file for this database partition. The path must end with a slash.

db2gRecoverStruct data structure specific parameters:

iSourceDBAliasLen
Specifies the length in bytes of the piSourceDBAlias parameter.

iUserNameLen
Specified the length in bytes of the piUsername parameter.

iPasswordLen
Specifies the length in bytes of the piPassword parameter.

iStopTimeLen
Specifies the length in bytes of the piStopTime parameter.

iOverflowLogPathLen
Specifies the length in bytes of the piOverflowLogPath parameter.

iHistoryFileLen
Specifies the length in bytes of the piHistoryFile parameter.

iComprLibraryLen
Input. Specifies the length in bytes of the name of the library specified in the piComprLibrary parameter. Set to zero if no library name is given.

Related tasks:

- "Using recover" on page 192

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "RECOVER DATABASE" on page 193
- "db2Rollforward - Roll forward a database" on page 177
- "db2Backup - Back up a database or table space" on page 76
- "db2Restore - Restore a database or table space" on page 115

Part 2. High availability

Chapter 6. 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 (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 refers to the simultaneous processing of a query on all participating nodes; each database partition 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.

You can implement failover capability through the high availability disaster recovery (HADR) database replication feature. HADR is a high availability solution that protects against data loss by replicating changes from a source database, called the primary database, to a target database, called the standby database.

Failover protection can also 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 also use suspended I/O support to quickly initialize the new database. The standby

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 standby database on the standby machine.

Failover support can also be provided through platform-specific software that you can add to your system. For example:

- Tivoli System Automation for Linux.

  For detailed information about Tivoli System Automation, see the whitepaper entitled "Highly Available DB2 Universal Database using Tivoli System Automation for Linux", which is available from the "DB2 Database for Linux, UNIX, and Windows and DB2 Connect Online Support" web site (http://www.ibm.com/software/data/pubs/papers/).
- 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 Database for Linux, UNIX, and Windows 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 paper which is available from the " DB2 Database for Linux, UNIX, and Windows Support" web site (http://www.ibm.com/software/data/pubs/papers/): "Implementing IBM DB2 Universal Database V8.1 Enterprise Server Edition with Microsoft Cluster Server".
- Sun Cluster, or VERITAS Cluster Server, for the Solaris operating system.

  For information about Sun Cluster, see the white paper entitled "DB2 Universal Database and High Availability on Sun Cluster 3.X", which is available from the "DB2 Database for Linux, UNIX, and Windows Support" web site (http://www.ibm.com/software/data/pubs/papers/). For information about VERITAS Cluster Server, see the white paper entitled "DB2 UDB and High Availability with VERITAS Cluster Server", which is available from the "IBM Support and downloads" Web site (http://www.ibm.com/support/docview.wss?uid=swg21045033).
- Multi-Computer/ServiceGuard, for Hewlett-Packard.

  For detailed information about HP MC/ServiceGuard, see the white paper which discusses IBM DB2 ESE V8.1 with HP MC/ServiceGuard High Availability Software, which is available from the "IBM DB2 Information Management Products for HP" web site (http://www.ibm.com/software/data/hp/).

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 physical machine within a cluster contains one or more logical nodes. 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.

The client should treat this IP address in the same way as a physical address or hostname. For example, when cataloguing the instance on the client you should use the server's transferable IP address or a hostname that resolves to this address.

Failover software can 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 might 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 can 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 database partition, or multiple logical nodes.

Related concepts:

- "Parallelism" in *Administration Guide: Planning*
- "Developing a backup and recovery strategy" on page 3
- "High availability disaster recovery overview" on page 221
- "High Availability Cluster Multi-Processing support" on page 275
- "Microsoft Cluster Server support" on page 281
- "High availability through online split mirror and suspended I/O support" on page 210
- "Cluster support for the Solaris operating system" on page 285
- "Sun Cluster 3.0 support" on page 287
- "VERITAS Cluster Server support" on page 290

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 (for example, 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 using modern disk subsystems such as ESS and EMC, or another remote mirroring technology. 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 disaster recovery overview" on page 221
- "High availability through online split mirror and suspended I/O support" on page 210

Related tasks:

- "Using a split mirror as a standby database" on page 213

Related reference:

- Appendix I, "User exit for database recovery," on page 409

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 database partitions simultaneously. You can suspend a subset of one or more database partitions to create split mirrors for performing offline backups. If the catalog partition is included in the subset, it must be the last database partition to be suspended.

In a partitioned database environment, the **db2inidb** command must be run on every database partition before the split image from any of the database partitions can be used. The tool can be run on all database partitions simultaneously using the **db2_all** command. If; however, you are using the RELOCATE USING option, you cannot use the **db2_all** command to run **db2inidb** on all of the database partitions simultaneously. A separate configuration file must be supplied for each database partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the **db2relocatedb** command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the **db2relocatedb** command only needs to be run once on that database partition.

Note: Ensure that the split mirror contains all containers and directories which comprise the database, including the volume directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.

Related tasks:

- "Using a split mirror as a backup image" on page 214
- "Using a split mirror as a standby database" on page 213
- "Using a split mirror to clone a database" on page 212

Related reference:

- "DBPATHS administrative view – Retrieve database paths" in *Administrative SQL Routines and Views*

Online split mirror handling

Using a split mirror to clone a database

Use the following procedure to create a clone database. Although you can write to clone databases, they are generally used for read-only activities such as running reports.

Restrictions:

You cannot back up a cloned database, restore the backup image on the original system, or roll forward through log files produced on the original system. You can use the AS SNAPSHOT option, but this provides only an instantaneous copy of the database at that time when the I/O is suspended; any other outstanding uncommited work will be rolled back after the db2inidb command is executed on the clone.

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.

   Note: Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Catalog the mirrored database on the secondary system.

   Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the RELOCATE USING option of the **db2inidb** command to accomplish this.
5. Start the database instance on the secondary system:
   ```
   db2start
   ```
6. Initialize the mirrored database on the secondary system:
   ```
   db2inidb database_alias as snapshot
   ```

   If required, specify the RELOCATE USING option of the db2inidb command to relocate the clone database:
   ```
   db2inidb database_alias as snapshot relocate using relocatedbcfg.txt
   ```

   where the relocatedbcfg.txt file contains the information required to relocate the database.

Notes:

a. 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.
b. The database directory (including the volume directory), the log directory, and the container directories must be moved to the desired location before you use the RELOCATE USING option.

Related concepts:

- "High availability through online split mirror and suspended I/O support" on page 210

Related tasks:

- "Using a split mirror as a backup image" on page 214
- "Using a split mirror as a standby database" on page 213

Related reference:

- "DBPATHS administrative view – Retrieve database paths" in *Administrative SQL Routines and Views*
- "db2relocatedb - Relocate database command" in *Command Reference*
- "SET WRITE command" in *Command Reference*

Using a split mirror as a standby database

Use the following procedure to create a split mirror of a database for use as a standby database. If a failure occurs on the primary database and crash recovery is necessary, you can use the standby database to take over for the primary 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.

   Note: Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Catalog the mirrored database on the secondary system.

   Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the RELOCATE USING option of the **db2inidb** command to accomplish this.

5. Start the database instance on the secondary system:
   ```
   db2start
   ```
6. Initialize the mirrored database on the secondary system by placing it in rollforward pending state:
   ```
   db2inidb database_alias as standby
   ```

   If required, specify the RELOCATE USING option of the db2inidb command to relocate the standby database:
   ```
   db2inidb database_alias as standby relocate using relocatedbcfg.txt
   ```

   where the relocatedbcfg.txt file contains the information required to relocate the database.

   Notes:

   a. 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.
   b. The database directory (including the volume directory), the log directory, and the container directories must be moved to the desired location before you use the RELOCATE USING option.
7. Set up a user exit program to retrieve the log files from the primary system.
8. Roll the database forward to the end of the logs or to a point-in-time.
9. 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.
10. To bring the standby database online issue the ROLLFORWARD command with the STOP option specified.

    Note: The logs from the primary database cannot be applied to the mirrored database once it has been taken out of rollforward pending state.

Related concepts:

- "High availability through online split mirror and suspended I/O support" on page 210

Related tasks:

- "Using a split mirror to clone a database" on page 212
- "Using a split mirror as a backup image" on page 214

Related reference:

- "DBPATHS administrative view – Retrieve database paths" in *Administrative SQL Routines and Views*
- "db2inidb - Initialize a mirrored database " on page 317
- "db2relocatedb - Relocate database command" in *Command Reference*
- "SET WRITE command" in *Command Reference*

Using a split mirror as a backup image

Use the following procedure to create a split mirror of a primary database for use as a backup image. This procedure can be used instead of performing backup database operations on the primary database.

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.

   Note: Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
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 210

Related tasks:

- "Using a split mirror to clone a database" on page 212
- "Using a split mirror as a standby database" on page 213

Related reference:

- "db2inidb - Initialize a mirrored database " on page 317
- "DBPATHS administrative view – Retrieve database paths" in *Administrative SQL Routines and Views*

Fault monitor facility for Linux and UNIX

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 database is running. That is, the *init* daemon monitors the Fault Monitor Coordinator (FMC), the FMC monitors the fault monitors and the fault monitors monitor the DB2 database system.

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 (that is, 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.

The health monitor and the fault monitor are tools that work on a single database instance. The health monitor uses *health indicators* to evaluate the health of specific aspects of database manager performance or database performance. A health indicator measures the health of some aspect of a specific class of database objects, such as a table space. Health indicators can be evaluated against specific criteria to determine the health of that class of database object. In addition, health indicators can generate alerts to notify you when an indicator exceeds a threshold or indicates a database object is in a non-normal state

By comparison, the fault monitor is solely responsible for keeping the instance it is monitoring up and running. If the DB2 instance it is monitoring terminates unexpectedly, the fault monitor restarts the instance. The fault monitor is not available on Windows.

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, type the following command from a DB2 database command window:

```
db2fm -i db2inst1 -T 100
```

To update the STOP_TIMEOUT value to 200 seconds for instance DB2INST1, type the following command:

```
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, type the following command:

```
db2fm -i db2inst1 -T 100/200
```

To turn on fault monitoring for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -f yes
```

To turn off fault monitoring for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -f no
```

Note: If the fault monitor registry file does not exist, the default values will be used.

To confirm that fault monitor is no longer running for DB2INST1, type the following command on UNIX systems:

```
ps -ef|grep -i fm
```

On Linux, type the following command:

```
ps auxw|grep -i fm
```

An entry that shows db2fmd and DB2INST1 indicates that the fault monitor is still running on that instance. To turn off the fault monitor, type the following command as the instance owner:

```
db2fm -i db2inst1 -D
```

You can prevent the FMC from being launched by using the DB2 Fault Monitor Controller Utility (FMCU). The FMCU must be run as root because it accesses the system's inittab file. To block the FMC from being run, type the following command as root:

```
db2fmcu -d
```

Note: If you apply a DB2 fix pack this will be reset so that the inittab will again be configured to include the FMC. To prevent the FMC from being launched after you have applied a fix pack, you must reissue the above command.

To reverse the `db2fmcu -d` command and reconfigure the inittab to include the FMC, type the following command:

```
db2fmcu -u -p ←installpath→
```

where ←installpath→ is the directory where db2fmcd is installed.

You can also enable FMC to automatically start the instance when the system is first booted. To enable this feature for instance DB2INST1, type the following command:

```
db2iauto -on db2inst1
```

To turn off the autostart behaviour, type the following command:

```
db2iauto -off db2inst1
```

You can also prevent fault monitor processes from being launched for a specific instances on the system by changing a field in the global registry record for the instance. To change the global registry field to disable fault monitors for instance DB2INST1, type the following command as root:

```
db2greg -updinstrec instancename=db2inst1!startatboot=0
```

To reverse this command and re-enable fault monitors for instance DB2INST1, type the following command as root:

```
db2greg -updinstrec instancename=db2inst1!startatboot=1"
```

Related reference:

- "db2fm - DB2 fault monitor " on page 219

db2fm - DB2 fault monitor

Controls the DB2 fault monitor daemon. You can use **db2fm** to configure the fault monitor.

This command is only available on UNIX operating systems.

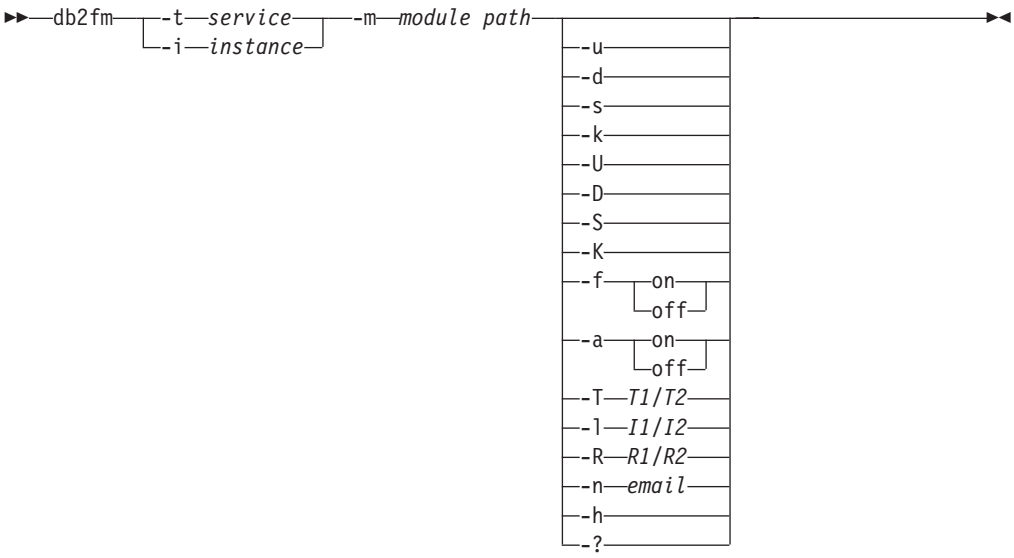
Authorization:

Authorization over the instance against which you are running the command.

Required connection:

None.

Command syntax:



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 instance 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. 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. 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. 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.

For example:

- -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.

Related concepts:

• "Fault monitor facility for Linux and UNIX" on page 215

Chapter 7. High availability disaster recovery (HADR)

High availability disaster recovery overview

DB2 database high availability disaster recovery (HADR) is a database replication feature that provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby. A database that does not use HADR is referred to as a standard database.

HADR might be your best option if most or all of your database requires protection, or if you perform DDL operations that must be automatically replicated on the standby database.

Applications can only access the current primary database. Updates to the standby database occur by rolling forward log data that is generated on the primary database and shipped to the standby database.

A partial site failure can be caused by a hardware, network, or software (DB2 database system or operating system) failure. Without HADR, a partial site failure requires restarting the database management system (DBMS) server that contains the database. The length of time it takes to restart the database and the server where it resides is unpredictable. It can take several minutes before the database is brought back to a consistent state and made available. With HADR, the standby database can take over in seconds. Further, you can redirect the clients that were using the original primary database to the standby database (new primary database) by using automatic client reroute or retry logic in the application.

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. Because HADR uses TCP/IP for communication between the primary and standby databases, they can be situated in different locations. For example, your primary database might be located at your head office in one city, while your standby database is located at your sales office in another city. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database with full DB2 functionality. After a takeover operation occurs, you can bring the original primary database back up and return it to its primary database status; this is known as failback.

With HADR, you can choose the level of protection you want from potential loss of data by specifying one of three synchronization modes: synchronous, near synchronous, or asynchronous.

After the failed original primary server is repaired, it can rejoin the HADR pair as a standby database if the two copies of the database can be made consistent. After the original primary database is reintegrated into the HADR pair as the standby database, you can switch the roles of the databases to enable the original primary database to once again be the primary database.

HADR is only one of several replication solutions offered in the DB2 product family. WebSphere® Information Integrator and the DB2 database system include SQL replication and Q replication solutions that can also be used, in some

configurations, to provide high availability. These functions maintain logically consistent copies of database tables at multiple locations. In addition, they provide flexibility and complex functionality such as support for column and row filtering, data transformation, updates to any copy of a table, and they can be used in partitioned database environments.

Related concepts:

- "Automatic client reroute and high availability disaster recovery (HADR)" on page 255
- "Database configuration for high availability disaster recovery (HADR)" on page 249
- "High availability through log shipping" on page 209
- "Restrictions for high availability disaster recovery (HADR)" on page 226
- "Synchronization modes for high availability disaster recovery (HADR)" on page 229
- "System requirements for high availability disaster recovery (HADR)" on page 222
- "Catchup state" in *Administration Guide: Planning*
- "Client reroute" in *Administration Guide: Planning*
- "Failback" in *Administration Guide: Planning*
- "Peer state" in *Administration Guide: Planning*

System requirements for high availability disaster recovery (HADR)

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following requirements for hardware, operating systems, and for the DB2 database system.

Recommendation: For better performance, use the same hardware and software for the system where the primary database resides and for the system where the standby database resides. If the system where the standby database resides has fewer resources than the system where the primary database resides, it is possible that the standby database will be unable to keep up with the transaction load generated by the primary database. This can cause the standby database to fall behind or the performance of the primary database to degrade. In a failover situation, the new primary database should have the resources to service the client applications adequately.

Hardware and operating system requirements:

Recommendation: Use identical host computers for the HADR primary and standby databases. That is, they should be from the same vendor and have the same architecture.

The operating system on the primary and standby databases should be the same version, including patches. You can violate this rule for a short time during a rolling upgrade, but take extreme caution.

A TCP/IP interface must be available between the HADR host machines, and a high-speed, high-capacity network is recommended.

DB2 database requirements:

The versions of the database systems for the primary and standby databases must be identical; for example, both must be either version 8 or version 9. During rolling upgrades, the modification level (for example, the fix pack level) of the database system for the standby database can be later than that of the primary database for a short while to test the new level. However, you should not keep this configuration for an extended period of time. The primary and standby databases will not connect to each other if the modification level of the database system for the primary database is later than that of the standby database.

The DB2 database software for both the primary and standby databases must have the same bit size (32 or 64 bit). Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If you create a table space on the primary database and log replay fails on the standby database because the containers are not available, the primary database does not receive an error message stating that the log replay failed.

To check for log replay errors, you must monitor the db2diag.log and the administration log on the standby database when you are creating new table spaces.

If a takeover operation occurs, the new table space that you created is not available on the new primary database. To recover from this situation, restore the table space on the new primary database from a backup image.

In the following example, table space MY_TABLESPACE is restored on database MY_DATABASE before it is used as the new primary database:

1. `db2 connect to my_database`
2. `db2 list tablespaces show detail`

   Note: Run the **db2 list tablespaces show detail** command to show the status of all table spaces and to obtain the table space ID number required for Step 5.
3. `db2 stop hadr on database my_database`
4. `db2 "restore database my_database tablespace (my_tablespace) online redirect"`
5. `db2 "set tablespace containers for my_tablespace_ID_# ignore rollforward container operations using (path '/my_new_container_path/')"`
6. `db2 "restore database my_database continue"`
7. `db2 rollforward database my_database to end of logs and stop tablespace "(my_tablespace)"`
8. `db2 start hadr on database my_database as primary`

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases.

Automatic storage databases are fully supported by HADR, including replication of the ALTER DATABASE statement with the ADD STORAGE ON clause. Similar to table space containers, the storage path must exist on both primary and standby.

The primary and standby databases must have the same database name. This means that they must be in different instances.

Redirected restore is not supported. That is, HADR does not support redirecting table space containers. However, database directory and log directory changes are supported. Table space containers created by relative paths will be restored to paths relative to the new database directory.

Buffer pool requirements:

Since buffer pool operations are also replayed on the standby database, it is important that the primary and standby databases have the same amount of memory.

Related concepts:

- "Restrictions for high availability disaster recovery (HADR)" on page 226
- "High availability disaster recovery overview" on page 221

Related tasks:

- "Performing a rolling upgrade in a high availability disaster recovery environment" on page 269

Installation and storage requirements for high availability disaster recovery

Automatic storage databases are fully supported by HADR, including replication of the ALTER DATABASE statement with the ADD STORAGE ON clause. Similar to table space containers, the storage path must exist on both primary and standby. Symbolic links can be used to create identical paths. The primary and standby databases can be on the same computer. Even though their database storage starts at the same path, they do not conflict because the actual directories used have instance names embedded in them (since the primary and standby databases must have the same database name, they must be in different instances). The storage path is formulated as `storage_path_name/inst_name/dbpart_name/db_name/tbsp_name/container_name`.

Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include: the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). If the database is enabled for automatic storage then the storage paths must be identical. This includes the path names and the amount of space on each that is devoted to the database. The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If the table space setup is not identical on the primary and standby databases, log replay on the standby database might encounter errors such as OUT OF SPACE or TABLE SPACE CONTAINER NOT FOUND. Similarly, if the databases are enabled for automatic storage and the storage paths are not identical, log records associated with the ADD STORAGE ON clause of the ALTER DATABASE statement will not be replayed. As a result, the existing storage paths might prematurely run out of space on the standby system and automatic storage table spaces will not be able to increase in size. If any of these situations occurs, the affected table space is put in rollforward pending state and is ignored in subsequent log replay. If a takeover operation occurs, the table space will not be available to applications.

If the problem is noticed on the standby system prior to a takeover then the resolution is to re-establish the standby database while addressing the storage issues. The steps to do this include:

- Deactivating the standby database.
- Dropping the standby database.
- Ensuring the necessary filesystems exist with enough free space for the subsequent restore and rollforward.
- Restoring the database at the standby system using a recent backup of the primary database (or, reinitialize using split mirror or flash copy with the db2inidb command). If the primary database is enabled for automatic storage then do not redefine the storage paths during the restore. Also, table space containers should not be redirected as part of the restore.
- Restarting HADR on the standby system.

However, if the problem is noticed with the standby database after a takeover has occurred (or if a choice was made to not address the storage issues until this time) then the resolution is based on the type of problem that was encountered.

If the database is enabled for automatic storage and space is not available on the storage paths associated with the standby database then follow these steps:

1. Make space available on the storage paths by extending the filesystems, or by removing unnecessary non-DB2 files on them.
2. Perform a table space rollforward to the end of logs.

In the case where the addition or extension of containers as part of log replay could not occur, if the necessary backup images and log file archives are available, you might be able to recover the table space by first issuing the SET TABLESPACE CONTAINERS statement with the IGNORE ROLLFORWARD CONTAINER OPERATIONS option and then issuing the ROLLFORWARD command.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases. Consequently, if the primary and standby databases are placed on the same computer, all table space containers must be defined with relative paths so that they map to different paths for primary and standby.

Installation requirements:

For HADR, instance paths should be the same on the primary and the standby databases. Using different instance paths can cause problems in some situations,

such as if an SQL stored procedure invokes a user-defined function (UDF) and the path to the UDF object code is expected to be on the same directory for both the primary and standby server.

Related concepts:

- "High availability disaster recovery overview" on page 221

Restrictions for high availability disaster recovery (HADR)

The following list is a summary of high availability disaster recovery (HADR) restrictions:

- HADR is not supported in a partitioned database environment.
- The primary and standby databases must have the same operating system version and the same version of the DB2 database system, except for a short time during a rolling upgrade.
- The DB2 database system release on the primary and standby databases must be the same bit size (32 or 64 bit).
- Reads on the standby database are not supported. Clients cannot connect to the standby database.
- Log archiving can only be performed by the current primary database.
- Self Tuning Memory Manager (STMM) can be run only on the current primary database.
- Backup operations are not supported on the standby database.
- Non-logged operations, such as changes to database configuration parameters and to the recovery history file, are not replicated to the standby database.
- Load operations with the COPY NO option specified are not supported.
- Use of Data Links is not supported.
- HADR does not support the use of raw i/o (direct disk access) for database log files. If HADR is started via the START HADR command, or the database is activated (restarted) with HADR configured, and raw logs are detected, the associated command will fail.

Related concepts:

- "High availability disaster recovery overview" on page 221
- "Non-replicated operations for high availability disaster recovery (HADR)" on page 233
- "Replicated operations for high availability disaster recovery (HADR)" on page 232
- "System requirements for high availability disaster recovery (HADR)" on page 222

Standby database states in high availability disaster recovery (HADR)

With the high availability disaster recovery (HADR) feature, when the standby database is started, it enters local catchup state and attempts to read the log files in its local log path. If it does not find a log file in the local log path and a log archiving method has been specified, the log file is retrieved using the specified method. After the log files are read, they are replayed on the standby database. During this time, a connection to the primary database is not required; however, if

a connection does not exist, the standby database tries to connect to the primary database. When the end of local log files is reached, the standby database enters remote catchup pending state.

The standby database remains in remote catchup pending state until a connection to the primary database is established, at which time the standby database enters remote catchup state. During this time, the primary database reads log data from its log path or by way of a log archiving method and sends the log files to the standby database. The standby database receives and replays the log data. When all of the log files on disk have been replayed by the standby database, the primary and standby systems enter peer state.

When in peer state, log pages are shipped to the standby database whenever the primary database flushes a log page to disk. The log pages are written to the local log files on the standby database to ensure that the primary and standby databases have identical log file sequences. The log pages can then be replayed on the standby database.

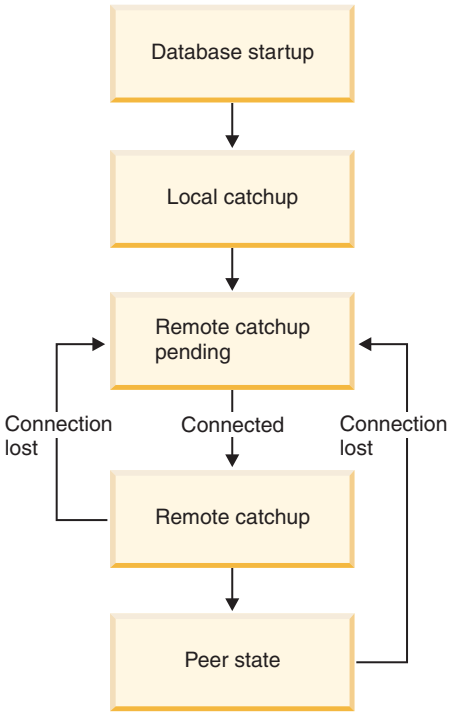


Figure 20. States of the standby database

You can see the state of the standby database by issuing the GET SNAPSHOT command with the DATABASE ON option. For example, if you have standby database MUSIC, you can issue the following command to see its state:

```
get snapshot for database on music
```

If the command is issued on the standby database, one of the standby database states is returned in the `State` field. If the query is issued on a primary database that is connected to a standby database, the state of the standby database is returned. If the primary database is not connected to a standby database, `disconnected` is returned.

The following output shows the HADR status section returned by the GET SNAPSHOT command:

```
HADR status

  Role                    = Primary
  State                   = Peer
  Synchronization mode    = Sync
  Connection status       = Connected, 11-03-2002 12:23:09.35092
  Heartbeat missed        = 0
  Local host              = host1.ibm.com
  Local service           = hadr_service
  Remote host             = host2.ibm.com
  Remote service          = hadr_service
  Remote instance         = dbinst2
  timeout(seconds)        = 120
  Primary log position(file, page, LSN) = S0001234.LOG, 12, 0000000000BB800C
  Standby log position(file, page, LSN) = S0001234.LOG, 12, 0000000000BB800C
  Log gap running average(bytes) = 8723
```

Notes:

1. If the connection between the primary and standby databases is lost during remote catchup or peer state, the standby database will enter remote catchup pending state.
2. Because the standby database writes the log files it receives to its local log path, you must not use a shared network or local file system as the log path for both the primary and standby databases. You will receive an error message if DB2 detects a shared log path.
3. To speed up the catchup process, you can use a shared log archive device. However, if the shared device is a serial device such as a tape drive, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations.
4. You can manually copy the primary database log files into the standby database log path to be used for local catchup. This must be done before you start the standby database, because when the end of the local log files is reached, the standby database will enter remote catchup pending state and will not try to access the log files again. Also, if the standby database enters remote catchup state, copying log files into its log path will interfere with the writing of local log files by the standby database. If more local log files become available after the standby database enters remote catchup pending state, you can shut down the standby database and restart it to ensure that it re-enters local catchup state.
5. When a log file is truncated, either as the result of an explicit log truncation, or as a result of stopping and restarting the primary database, the primary moves to the beginning of the next log file. The standby, however, stays at the end of the last log file. As a result, the HADR snapshot will show a log gap. As soon as the primary writes any log, the log will be replicated and the standby will update its log position.

Related concepts:

- "Log file management" on page 46
- "Log file management through log archiving" on page 49
- "Synchronization modes for high availability disaster recovery (HADR)" on page 229

Related tasks:

- "Performing a rolling upgrade in a high availability disaster recovery environment" on page 269

- "Reintegrating a database after a takeover operation" on page 268
- "Switching database roles in high availability disaster recovery (HADR)" on page 259
- "Performing an HADR failover operation" on page 261

Related reference:

- "GET SNAPSHOT command" in *Command Reference*
- "Configuration parameters for database logging" on page 37
- Appendix I, "User exit for database recovery," on page 409
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*
- "logarchmeth2 - Secondary log archive method configuration parameter" in *Performance Guide*

Synchronization modes for high availability disaster recovery (HADR)

With high availability disaster recovery (HADR), you can specify one of three synchronization modes to choose your preferred level of protection from potential loss of data. The synchronization mode indicates how log writing is managed between the primary and standby databases. These modes apply only when the primary and standby databases are in peer state.

Use the HADR_SYNCMODE configuration parameter to set the synchronization mode. Valid values are:

SYNC (synchronous)

This mode provides the greatest protection against transaction loss, and using it results in the longest transaction response time among the three modes.

In this mode, log writes are considered successful only when logs have been written to log files on the primary database and when the primary database has received acknowledgement from the standby database that the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

If the standby database crashes before it can replay the log records, the next time it starts it can retrieve and replay them from its local log files. If the primary database fails, a failover to the standby database guarantees that any transaction that has been committed on the primary database has also been committed on the standby database. After the failover operation, when the client reconnects to the new primary database, there can be transactions committed on the new primary database that were never reported as committed to the original primary. This occurs when the primary database fails before it processes an acknowledgement message from the standby database. Client applications should consider querying the database to determine whether any such transactions exist.

If the primary database loses its connection to the standby database, the databases are no longer considered to be in peer state and transactions will not be held back waiting for acknowledgement from the standby database. If the failover operation is performed when the databases are disconnected, there is no guarantee that all of the transactions committed on the primary database will appear on the standby database.

If the primary database fails when the databases are in peer state, it can rejoin the HADR pair as a standby database after a failover operation. Because a transaction is not considered to be committed until the primary database receives acknowledgement from the standby database that the logs have also been written to log files on the standby database, the log sequence on the primary will be the same as the log sequence on the standby database. The original primary database (now a standby database) just needs to catch up by replaying the new log records generated on the new primary database since the failover operation.

If the primary database is not in peer state when it fails, its log sequence might be different from the log sequence on the standby database. If a failover operation has to be performed, the log sequence on the primary and standby databases might be different because the standby database starts its own log sequence after the failover. Because some operations cannot be undone (for example, dropping a table), it is not possible to revert the primary database to the point in time when the new log sequence was created. If the log sequences are different and you issue the START HADR command with the STANDBY option on the original primary, you will receive a message that the command was successful. However, this message is issued before reintegration is attempted. If reintegration fails, pair validation messages will be issued to the administration log and the diagnostics log on both the primary and the standby. The reintegrated standby will remain the standby, but the primary will reject the standby during pair validation causing the standby database to shut down. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

NEARSYNC (near synchronous)

While this mode has a shorter transaction response time than synchronous mode, it also provides slightly less protection against transaction loss.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and when the primary database has received acknowledgement from the standby system that the logs have also been written to main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred to nonvolatile storage all of the log data that it has received.

If the standby database crashes before it can copy the log records from memory to disk, the log records will be lost on the standby database. Usually, the standby database can get the missing log records from the primary database when the standby database restarts. However, if a failure on the primary database or the network makes retrieval impossible and a failover is required, the log records will never appear on the standby database, and transactions associated with these log records will never appear on the standby database.

If transactions are lost, the new primary database is not identical to the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records (because both the primary and standby databases have failed), the log sequences on the primary and standby databases will be different and attempts to restart the original primary database as a standby database without first performing a restore operation will fail. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

ASYNC (asynchronous)

This mode has the highest chance of transaction loss if the primary system fails. It also has the shortest transaction response time among the three modes.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and have been delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater possibility of log records being lost if a failover occurs in asynchronous mode, there is also a greater possibility that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

Related concepts:

- "Standby database states in high availability disaster recovery (HADR)" on page 226

Related tasks:

- "Reintegrating a database after a takeover operation" on page 268

Related reference:

- "hadr_syncmode - HADR synchronization mode for log write in peer state configuration parameter" in *Performance Guide*

Replicated operations for high availability disaster recovery (HADR)

In high availability disaster recovery (HADR), the following operations are replicated from the primary to the standby database:

- Data definition language (DDL)
- Data manipulation language (DML)
- Buffer pool operations
- Table space operations
- Online reorganization
- Offline reorganization
- Metadata for stored procedures and user defined functions (UDF) (but not the related object or library files)

During an online reorganization, all operations are logged in detail. As a result, HADR can replicate the operation without the standby database falling further behind than it would for more typical database updates. However, this behavior can potentially have a large impact on the system because of the large number of log records generated.

While offline reorganizations are not logged as extensively as online reorganizations, operations are typically logged per hundreds or thousands of affected rows. This means that the standby database could fall behind because it waits for each log record and then replays many updates at once. If the offline reorganization is non-clustered, a single log record is generated after the entire reorganization operation. This mode has the greatest impact on the ability of the standby database to keep up with the primary database. The standby database will perform the entire reorganization after it receives the log record from the primary database.

HADR does not replicate stored procedure and UDF object and library files. You must create the files on identical paths on both the primary and standby databases. If the standby database cannot find the referenced object or library file, the stored procedure or UDF invocation will fail on the standby database.

Related concepts:

- "Database configuration for high availability disaster recovery (HADR)" on page 249
- "Non-replicated operations for high availability disaster recovery (HADR)" on page 233
- "Restrictions for high availability disaster recovery (HADR)" on page 226
- "Index reorganization" in *Performance Guide*
- "Table reorganization" in *Performance Guide*

Non-replicated operations for high availability disaster recovery (HADR)

High availability disaster recovery (HADR) uses database logs to replicate data to the standby database. Non-logged operations are allowed on the primary database, but not replicated to the standby database. The following are examples of cases in which operations on the primary database are not replicated to the standby database:

- Tables created with the NOT LOGGED INITIALLY option specified are not replicated. Attempts to access such tables after an HADR standby database takes over as the primary database will result in an error.
- BLOBs and CLOBs that are larger than 1GB cannot be logged, so they cannot be replicated. Non-logged BLOBs and CLOBs are not replicated. However, the space for them will be allocated on the standby database. The data for the LOB column will be binary zeroes. All logged BLOBs and CLOBs are replicated.
- Datalinks are not supported in HADR databases. If the START HADR or the ACTIVATE DATABASE command is issued, or if a first client connection brings up a database that is in an HADR role (primary or standby), and the DATALINKS database manager configuration parameter is set to YES, the operation will fail. To use the HADR database, set the DATALINKS configuration parameter to NO, shut down the instance, then restart it.

  Note: Existing datalink columns are not affected when a database is converted from a standard database to a primary or standby database. Even if the DATALINKS configuration parameter is set to NO, you can still create new datalink columns in an HADR database. However, you cannot insert, update or select datalink columns.
- Updates to database configuration using the UPDATE DATABASE CONFIGURATION and UPDATE DATABASE MANAGER CONFIGURATION commands are not replicated.
- Database configuration and database manager configuration parameters are not replicated.
- For user-defined functions (UDFs), changes to objects external to the database (such as related objects and library files) are not replicated. They will need to be setup on the standby via other means.
- The recovery history file (db2rhist.asc), and changes to it, are not automatically shipped from the primary database to the standby database.

  You can place an initial copy of the history file (obtained from the backup image of the primary) on the standby database by issuing the RESTORE DATABASE command with the REPLACE HISTORY FILE option:

  ```
  RESTORE DB KELLY REPLACE HISTORY FILE
  ```

  After HADR is initialized and subsequent backup activities take place on the primary database, the history file on the standby database will become out of date. However, a copy of the history file is stored in each backup image. You can update the history file on the standby by extracting the history file from a backup image using the following command:

  ```
  RESTORE DB KELLY HISTORY FILE
  ```

  Do not use regular operating system commands to copy the history file in the database directory from the primary database to the standby database. The history file can become corrupted if the primary is updating the files when the copy is made.

If a takeover operation occurs and the standby database has an up-to-date history file, backup and restore operations on the new primary will generate new records in the history file and blend seamlessly with the records generated on the original primary. If the history file is out of date or has missing entries, an automatic incremental restore might not be possible; instead, a manual incremental restore operation will be required.

Related concepts:

- "Database configuration for high availability disaster recovery (HADR)" on page 249
- "Replicated operations for high availability disaster recovery (HADR)" on page 232
- "Restrictions for high availability disaster recovery (HADR)" on page 226
- "Understanding the recovery history file" on page 56
- "Index reorganization" in *Performance Guide*
- "Table reorganization" in *Performance Guide*

High availability disaster recovery (HADR) commands overview

There are three high availability disaster recover (HADR) commands used to manage HADR:

- Start HADR
- Stop HADR
- Takeover HADR

To invoke these commands, use the command line processor or the administrative API. You can also invoke these commands using the GUIs available from the Manage High Availability Disaster Recovery window in the Control Center. To open the Manage High Availability Disaster Recovery window from the Control Center, right-click a database and click High Availability Disaster Recovery-—>Manage.

Issuing the START HADR command with either the AS PRIMARY or AS STANDBY option changes the database role to the one specified if the database is not already in that role. This command also activates the database, if it is not already activated.

The STOP HADR command changes an HADR database (either primary or standby) into a standard database. Any database configuration parameters related to HADR remain unchanged so that the database can easily be reactivated as an HADR database.

The TAKEOVER HADR command, which you can issue on the standby database only, changes the standby database to a primary database. When you do not specify the BY FORCE option, the primary and standby databases switch roles. When you do specify the BY FORCE option, the standby database unilaterally switches to become the primary database. In this case, the standby database attempts to stop transaction processing on the old primary database. However, there is no guarantee that transaction processing will stop. Use the BY FORCE option to force a takeover operation for failover conditions only. To whatever extent possible, ensure that the current primary has definitely failed, or shut it down yourself, prior to issuing the TAKEOVER HADR command with the BY FORCE option.

HADR database role switching

A database can be switched between primary and standard roles dynamically and repeatedly. When the database is either online or offline, you can issue both the START HADR command with the AS PRIMARY option and the STOP HADR command.

You can switch a database between standby and standard roles statically. You can do so repeatedly only if the database remains in rollforward pending state. You can issue the START HADR command with the AS STANDBY option to change a standard database to standby while the database is offline and in rollforward pending state. Use the STOP HADR command to change a standby database to a standard database while the database is offline. The database remains in rollforward pending state after you issue the STOP HADR command. Issuing a subsequent START HADR command with the AS STANDBY option returns the database to standby. If you issue the ROLLFORWARD DATABASE command with the STOP option after stopping HADR on a standby database, you cannot bring it back to standby. Because the database is out of rollforward pending state, you can use it as a standard database. This is referred to as taking a snapshot of the standby database. After changing an existing standby database into a standard database, consider creating a new standby database for high availability purposes.

To switch the role of the primary and standby databases, perform a takeover operation without using the BY FORCE option.

To change the standby to primary unilaterally (without changing the primary to standby), use forced takeover. Subsequently, you might be able to reintegrate the old primary as a new standby.

HADR role is persistent. Once an HADR role is established, it remains with the database, even through repeated stopping and restarting of the DB2 instance or deactivation and activation of the DB2 database.

Starting the standby is asynchronous

When you issue the START HADR command with the AS STANDBY option, the command returns as soon as the relevant engine dispatchable units (EDUs) are successfully started. The command does not wait for the standby to connect to the primary database. In contrast, the primary database is not considered started until it connects to a standby database (with the exception of when the START HADR command is issued on the primary with the BY FORCE option). If the standby database encounters an error, such as the connection being rejected by the primary database, the START HADR command with the AS STANDBY option might have already returned successfully. As a result, there is no user prompt to which HADR can return an error indication. The HADR standby will write a message to the DB2 diagnostic log and shut itself down. You should monitor the status of the HADR standby to ensure that it successfully connects with the HADR primary.

Replay errors, which are errors that the standby encounters while replaying log records, can also bring down the standby database. These errors might occur, for example, when there is not enough memory to create a buffer pool, or if the path is not found while creating a table space. You should continuously monitor the status of the standby database.

Do not run HADR commands from a client using a database alias enabled for client reroute

When automatic client reroute is set up, the database server has a predefined alternate server so that client applications can switch between working with either the original database server or the alternative server with only minimal interruption of the work. In such an environment, when a client connects to the database via TCP, the actual connection can go to either the original database or to the alternate database. HADR commands are implemented to identify the target database through regular client connection logic. Consequently, if the target database has an alternative database defined, it is difficult to determine the database on which the command is actually operating. Although an SQL client does not need to know which database it is connecting to, HADR commands must be applied on a specific database. To accommodate this limitation, HADR commands should be issued locally on the server machine so that client reroute is bypassed (client reroute affects only TCP/IP connections).

HADR commands must be run on a server with a valid license

The START HADR, STOP HADR, and TAKEOVER HADR commands require that a valid HADR license has been installed on the server where the command is executed. If the license is not present, these commands will fail and return a command-specific error code (SQL01767N, SQL01769N, or SQL01770N, respectively) along with a reason code of 98. To correct the problem, either install a valid HADR license using **db2licm**, or install a version of the server that contains a valid HADR license as part of its distribution.

Related concepts:

- "Automatic client reroute description and setup" in *Administration Guide: Implementation*
- "Automatic client reroute and high availability disaster recovery (HADR)" on page 255
- "High availability disaster recovery overview" on page 221

Related tasks:

- "Initializing high availability disaster recovery (HADR)" on page 238
- "Performing an HADR failover operation" on page 261
- "Stopping high availability disaster recovery (HADR)" on page 244
- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Related reference:

- "db2HADRStart - Start high availability disaster recovery (HADR) operations" on page 242
- "db2HADRStop - Stop high availability disaster recovery (HADR) operations" on page 247
- "db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database" on page 266
- "Command line processor features" in *Command Reference*
- "START HADR" on page 240
- "STOP HADR" on page 246
- "TAKEOVER HADR" on page 264

High availability disaster recovery (HADR) management

High availability disaster recovery management involves configuring and maintaining the status of your HADR system.

Managing HADR includes such tasks as:

- Initializing high availability disaster recovery (HADR)
- Stopping high availability disaster recovery (HADR)
- Switching database roles in high availability disaster recovery (HADR)
- Performing an HADR failover operation
- Monitoring high availability disaster recovery (HADR)
- Checking or altering database configuration parameters related to HADR. See: Configuring DB2 with configuration parameters
- Cataloging an HADR database (if required). See: Cataloging a database

You can manage HADR using the following methods:

- Command line processor
- Control Center GUI tools
- DB2 administrative API

Related concepts:

- "Automatic client reroute and high availability disaster recovery (HADR)" on page 255
- "Cluster managers and high availability disaster recovery (HADR)" on page 258
- "Database activation and deactivation in high availability disaster recovery (HADR)" on page 254
- "High availability disaster recovery (HADR) performance" on page 271
- "Index logging and high availability disaster recovery (HADR)" on page 256
- "Monitoring high availability disaster recovery (HADR)" on page 270

Related tasks:

- "Performing an HADR failover operation" on page 261
- "Configuring DB2 with configuration parameters" in *Performance Guide*
- "Cataloging a database" in *Administration Guide: Implementation*
- "Initializing high availability disaster recovery (HADR)" on page 238
- "Performing a rolling upgrade in a high availability disaster recovery environment" on page 269
- "Stopping high availability disaster recovery (HADR)" on page 244
- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Related reference:

- "db2HADRStart - Start high availability disaster recovery (HADR) operations" on page 242
- "db2HADRStop - Stop high availability disaster recovery (HADR) operations" on page 247
- "db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database" on page 266
- "START HADR" on page 240

- "STOP HADR" on page 246
- "TAKEOVER HADR" on page 264

Initializing high availability disaster recovery (HADR)

Use the following procedure to set up and initialize the primary and standby databases for high availability disaster recovery (HADR).

Procedure:

HADR can be initialized through the command line processor (CLP), the Set Up High Availability Disaster Recovery (HADR) wizard in the Control Center, or by calling the db2HADRStart API.

To use the CLP to initialize HADR on your system for the first time:

1. Determine the host name, host IP address, and the service name or port number for each of the HADR databases.

   If a host has multiple network interfaces, ensure that the HADR host name or IP address maps to the intended one. You need to allocate separate HADR ports in /etc/services for each protected database. These cannot be the same as the ports allocated to the instance. The host name can only map to one IP address.

   Note: The instance names for the primary and standby databases do not have to be the same.
2. Create the standby database by restoring a backup image or by initializing a split mirror, based on the existing database that is to be the primary.

   In the following example, the BACKUP DATABASE and RESTORE DATABASE commands are used to initialize database SOCKS as a standby database. In this case, an NFS mounted file system is accessible at both sites.

   Issue the following command at the primary database:

   ```
   backup db socks to /nfs1/backups/db2/socks
   ```

   Issue the following command at the standby database:

   ```
   restore db socks from /nfs1/backups/db2/socks replace history file
   ```

   The following example illustrates how to use the **db2inidb** utility to initialize the standby database using a split mirror of the primary database. This procedure is an alternative to the backup and restore procedure illustrated above.

   Issue the following command at the standby database:

   ```
   db2inidb socks as standby
   ```

   Notes:

   a. The database names for the primary and standby databases must be the same.

   b. It is recommended that you do not issue the ROLLFORWARD DATABASE command on the standby database after the restore operation or split mirror initialization. The results of using a rollforward operation might differ slightly from replaying the logs using HADR on the standby database. If the databases are not identical, issuing the START HADR command with the AS STANDBY option will fail.

   c. When using the RESTORE DATABASE command, it is recommended that the REPLACE HISTORY FILE option is used.

d. When creating the standby database using the RESTORE DATABASE command, you must ensure that the standby remains in rollforward mode. This means that you cannot issue the ROLLFORWARD DATABASE command with either the COMPLETE option or the STOP option. An error will be returned if the START HADR command with the AS STANDBY option is attempted on the database after rollforward is stopped.

e. The following RESTORE DATABASE command options should be avoided when setting up the standby database: TABLESPACE, INTO, REDIRECT, and WITHOUT ROLLING FORWARD.

f. When setting up the standby database using the **db2inidb** utility, do not use the SNAPSHOT or MIRROR options. You can specify the RELOCATE USING option to change one or more of the following configuration attributes: instance name, log path, and database path. However, you must not change the database name or the table space container paths.

3. Set the HADR configuration parameters on the primary and standby databases.

   Note: It is very important that you set the following configuration parameters after the standby databases has been created:
   - HADR_LOCAL_HOST
   - HADR_LOCAL_SVC
   - HADR_REMOTE_HOST
   - HADR_REMOTE_SVC
   - HADR_REMOTE_INST

   If they are set prior to creating the standby database, the settings on the standby database will reflect what is set on the primary database.

4. Connect to the standby instance and start HADR on the standby database, as in the following example:

   ```
   START HADR ON DB SOCKS AS STANDBY
   ```

   Note: Usually, the standby database is started first. If you start the primary database first, this startup procedure will fail if the standby database is not started within the time period specified by the HADR_TIMEOUT database configuration parameter.

5. Connect to the primary instance and start HADR on the primary database, as in the following example:

   ```
   START HADR ON DB SOCKS AS PRIMARY
   ```

6. HADR is now started on the primary and standby databases.

To open the Set Up High Availability Disaster Recovery (HADR) Databases wizard:

1. From the Control Center expand the object tree until you find the database for which you want to configure HADR.
2. Right-click the database and click High Availability Disaster Recovery → Set Up in the pop-up menu. The Set Up High Availability Disaster Recovery Databases wizard opens.

Additional information is provided through the contextual help facility within the Control Center.

Note: You can start HADR within the Set Up High Availability Disaster Recovery Databases wizard, or you can just use the wizard to initialize HADR, then start it at another time. To open the Start HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Click Start HADR. The Start HADR window opens.

Related concepts:

- "Database configuration for high availability disaster recovery (HADR)" on page 249
- "High availability disaster recovery overview" on page 221

Related tasks:

- "Stopping high availability disaster recovery (HADR)" on page 244
- "Using a split mirror as a backup image" on page 214
- "Using restore" on page 90

Related reference:

- "START HADR" on page 240
- "db2HADRStart - Start high availability disaster recovery (HADR) operations" on page 242

START HADR

Starts HADR operations for a database.

Authorization:

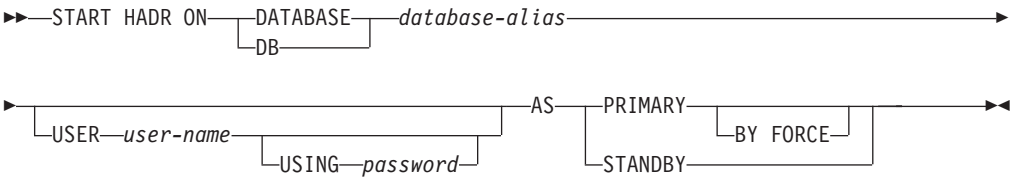
One of the following:

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

Required connection:

Instance. The command establishes a database connection if one does not exist, and closes the database connection when the command completes.

Command syntax:



Command parameters:

DATABASE *database-alias*
Identifies the database on which HADR operations are to start.

USER *user-name*
Identifies the user name under which the HADR operations are to be started.

USING *password*
The password used to authenticate *user-name*.

AS PRIMARY
Specifies that HADR primary operations are to be started on the database.

BY FORCE
Specifies that the HADR primary database will not wait for the standby database to connect to it. After a start BY FORCE, the primary database will still accept valid connections from the standby database whenever the standby later becomes available. When BY FORCE is used, the database will perform crash recovery if necessary, regardless of the value of database configuration parameter AUTORESTART. Other methods of starting a primary database (such as non-forced **START HADR** command, **ACTIVATE DATABASE** command, or client connection) will respect the AUTORESTART setting.

Caution: Use the **START HADR** command with the AS PRIMARY BY FORCE option with caution. If the standby database has been changed to a primary and the original primary database is restarted by issuing the **START HADR** command with the AS PRIMARY BY FORCE option, both copies of your database will be operating independently as primaries. (This is sometimes referred to as *split brain* or *dual primary*.) In this case, each primary database can accept connections and perform transactions, and neither receives and replays the updates made by the other. As a result, the two copies of the database will become inconsistent with each other.

AS STANDBY
Specifies that HADR standby operations are to be started on the database. The standby database will attempt to connect to the HADR primary database until a connection is successfully established, or until the connection attempt is explicitly rejected by the primary. (The connection might be rejected by the primary database if an HADR configuration parameter is set incorrectly or if the database copies are inconsistent, both conditions for which continuing to retry the connection is not appropriate.)

Usage notes:

The following table shows database behavior in various conditions:

Database status	Behavior upon START HADR command with the AS PRIMARY option	Behavior upon START HADR command with the AS STANDBY option
Inactive standard database	Activated as HADR primary database.	Database starts as an standby database if it is in rollforward-pending mode (which can be the result of a restore or a split mirror) or in rollforward in-progress mode. Otherwise, an error is returned.
Active standard database	Database enters HADR primary role.	Error message returned.

Database status	Behavior upon START HADR command with the AS PRIMARY option	Behavior upon START HADR command with the AS STANDBY option
Inactive primary database	Activated as HADR primary database.	After a failover, this reintegrates the failed primary into the HADR pair as the new standby database. Some restrictions apply.
Active primary database	Warning message issued.	Error message returned.
Inactive standby database	Error message returned.	Starts the database as the standby database.
Active standby database	Error message returned.	Warning message issued.

When issuing the **START HADR** command, the corresponding error codes might be generated: SQL01767N, SQL01769N, or SQL01770N with a reason code of 98. The reason code indicates that there is no installed license for HADR on the server where the command was issued. To correct the problem, install a valid HADR license using the **db2licm** or install a version of the server that contains a valid HADR license as part of its distribution.

Related tasks:

- "Initializing high availability disaster recovery (HADR)" on page 238

db2HADRStart - Start high availability disaster recovery (HADR) operations

Starts HADR operations on a database.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. The API establishes a database connection if one does not exist, and closes the database connection when the API completes.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HADRStart (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRStartStruct
{
```

```
   char *piDbAlias;
   char *piUserName;
   char *piPassword;
   db2Uint32 iDbRole;
   db2Uint16 iByForce;
} db2HADRStartStruct;

SQL_API_RC SQL_API_FN
  db2gHADRStart (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRStartStruct
{
   char *piDbAlias;
   db2Uint32 iAliasLen;
   char *piUserName;
   db2Uint32 iUserNameLen;
   char *piPassword;
   db2Uint32 iPasswordLen;
   db2Uint32 iDbRole;
   db2Uint16 iByForce;
} db2gHADRStartStruct;
```

db2HADRStart API parameters:

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

pParmStruct
Input. A pointer to the db2HADRStartStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2HADRStartStruct data structure parameters:

piDbAlias
Input. A pointer to the database alias.

piUserName
Input. A pointer to the user name under which the command will be executed.

piPassword
Input. A pointer to a string containing the password.

iDbRole
Input. Specifies which HADR database role should be started on the specified database. Valid values are:

DB2HADR_DB_ROLE_PRIMARY
Start HADR operations on the database in the primary role.

DB2HADR_DB_ROLE_STANDBY
Start HADR operations on the database in the standby role.

iByForce
Input. This argument is ignored if the iDbRole parameter is set to DB2HADR_DB_ROLE_STANDBY. Valid values are:

DB2HADR_NO_FORCE
Specifies that HADR is started on the primary database only if a standby database connects to it within a prescribed time limit.

DB2HADR_FORCE
Specifies that HADR is to be started by force, without waiting for the standby database to connect to the primary database.

db2gHADRStartStruct data structure specific parameters:

iAliasLen
Input. Specifies the length in bytes of the database alias.

iUserNameLen
Input. Specifies the length in bytes of the user name.

iPasswordLen
Input. Specifies the length in bytes of the password.

Related tasks:

- "Initializing high availability disaster recovery (HADR)" on page 238

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "db2HADRStop - Stop high availability disaster recovery (HADR) operations" on page 247
- "db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database" on page 266

Stopping high availability disaster recovery (HADR)

Use the STOP HADR command to stop high availability disaster recovery (HADR) operations on the primary or standby database. You can choose to stop HADR on one or both of the databases. If you are performing maintenance on the standby system, you only need to stop HADR on the standby database. If you want to stop using HADR completely, you can stop HADR on both databases.

Warning: If you want to stop the specified database but you still want it to maintain its role as either an HADR primary or standby database, do not issue the STOP HADR command. If you issue the STOP HADR command the database will become a standard database and might require reinitialization in order to resume operations as an HADR database. Instead, issue the DEACTIVATE DATABASE command.

Restrictions:

You can issue the STOP HADR command against a primary or a standby database only. If you issue this command against a standard database an error will be returned.

Procedure:

You can stop HADR by using the command line processor (CLP), the Manage High Availability Disaster Recovery (HADR) window in the Control Center, or the **db2HADRStop** application programming interface (API).

To use the CLP to stop HADR operations on the primary or standby database, issue the STOP HADR command on the database where you want to stop HADR operations.

In the following example, HADR operations are stopped on database SOCKS:

```
STOP HADR ON DATABASE SOCKS
```

If you issue this command against an inactive primary database, the database switches to a standard database and remains offline.

If you issue this command against an inactive standby database the database switches to a standard database, is placed in rollforward pending state, and remains offline.

If you issue this command on an active primary database, logs stop being shipped to the standby database and all HADR engine dispatchable units (EDUs) are shut down on the primary database. The database switches to a standard database and remains online. Transaction processing can continue. You can issue the START HADR command with the AS PRIMARY option to switch the role of the database back to primary database.

If you issue this command on an active standby database, an error message is returned, indicating that you must deactivate the standby database before attempting to convert it to a standard database.

To open the Stop HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Click Stop HADR. The Stop HADR window opens.
3. If you want to stop HADR on one database only, clear the check box for the other database.
4. If only one database is started (either the primary database or the standby database), the name of that database is displayed in the Stop HADR window.
5. Click OK. The window closes. A progress indicator might open to indicate when the command is running. When it completes, you will get a notification indicating whether or not it is successful.

Additional information is provided through the contextual help facility within the Control Center.

Related concepts:

- “High availability disaster recovery overview” on page 221
- “High availability disaster recovery (HADR) commands overview” on page 234

Related tasks:

- “Initializing high availability disaster recovery (HADR)” on page 238

Related reference:

- “db2HADRStart - Start high availability disaster recovery (HADR) operations” on page 242

- "db2HADRStop - Stop high availability disaster recovery (HADR) operations" on page 247
- "START HADR" on page 240
- "STOP HADR" on page 246

STOP HADR

Stops HADR operations for a database.

Authorization:

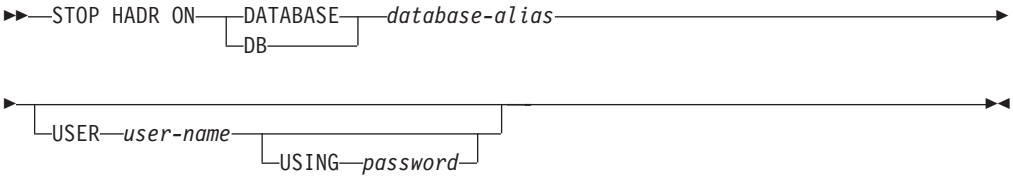
One of the following:

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

Required connection:

Instance. The command establishes a database connection if one does not exist, and closes the database connection when the command completes.

Command syntax:



Command parameters:

DATABASE *database-alias*
Identifies the database on which HADR operations are to stop.

USER *user-name*
Identifies the user name under which the HADR operations are to be stopped.

USING *password*
The password used to authenticate *user-name*.

Usage notes:

The following table shows database behavior in various conditions:

Database status	Behavior upon STOP HADR command
Inactive standard database	Error message returned.
Active standard database	Error message returned.
Inactive primary database	Database role changes to standard. Database configuration parameter *hadr_db_role* is updated to STANDARD. Database remains offline. At the next restart, enters standard role.

Database status	Behavior upon STOP HADR command
Active primary database	Stops shipping logs to the HADR standby database and shuts down all HADR EDUs on the HADR primary database. Database role changes to standard and database remains online. Database remains in standard role until an explicit START HADR command with the AS PRIMARY option is issued. Open sessions and transactions are not affected by the STOP HADR command. You can repeatedly issue STOP HADR and START HADR commands while the database remains online. These commands take effect dynamically.
Inactive standby database	Database role changes to standard. Database configuration parameter *hadr_db_role* is updated to STANDARD. Database remains offline. Database is put into rollforward pending mode.
Active standby database	Error message returned: Deactivate the standby database before attempting to convert it to a standard database.

When issuing the **STOP HADR** command, the corresponding error codes might be generated: SQL01767N, SQL01769N, or SQL01770N with a reason code of 98. The reason code indicates that there is no installed license for HADR on the server where the command was issued. To correct the problem, install a valid HADR license using the **db2licm** or install a version of the server that contains a valid HADR license as part of its distribution.

Related tasks:

- "Stopping high availability disaster recovery (HADR)" on page 244

db2HADRStop - Stop high availability disaster recovery (HADR) operations

Stops HADR operations on a database.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. The API establishes a database connection if one does not exist, and closes the database connection when the API completes.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HADRStop (
       db2Uint32 versionNumber,
       void * pParmStruct,
```

```
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRStopStruct
{
   char *piDbAlias;
   char *piUserName;
   char *piPassword;
} db2HADRStopStruct;

SQL_API_RC SQL_API_FN
  db2gHADRStop (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRStopStruct
{
   char *piDbAlias;
   db2Uint32 iAliasLen;
   char *piUserName;
   db2Uint32 iUserNameLen;
   char *piPassword;
   db2Uint32 iPasswordLen;
} db2gHADRStopStruct;
```

db2HADRStop API parameters:

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

pParmStruct
Input. A pointer to the db2HADRStopStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2HADRStopStruct data structure parameters:

piDbAlias
Input. A pointer to the database alias.

piUserName
Input. A pointer to the user name under which the command will be executed.

piPassword
Input. A pointer to a string containing the password.

db2gHADRStopStruct data structure specific parameters:

iAliasLen
Input. Specifies the length in bytes of the database alias.

iUserNameLen
Input. Specifies the length in bytes of the user name.

iPasswordLen
Input. Specifies the length in bytes of the password.

Related tasks:

- "Stopping high availability disaster recovery (HADR)" on page 244

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "db2HADRStart - Start high availability disaster recovery (HADR) operations" on page 242
- "db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database" on page 266

Database configuration for high availability disaster recovery (HADR)

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your database configuration meets the following requirements.

Recommendation: To the extent possible, the database configuration parameters and database manager configuration parameters should be identical on the systems where the primary and standby databases reside. If the configuration parameters are not properly set on the standby database the following problems might occur:

- Error messages might be returned on the standby database while replaying the log files that were shipped from the primary database.
- After a takeover operation, the new primary database will not be able to handle the workload, resulting in performance problems or in applications receiving error messages they were not receiving when they were connected to the original primary database.

Changes to the configuration parameters on the primary database are not automatically propagated to the standby database and must be made manually on the standby database. For dynamic configuration parameters, changes will take effect without shutting down and restarting the database management system (DBMS) or the database. For non-dynamic configuration parameters, changes will take effect after the standby database is restarted.

Size of log files configuration parameter on the standby database:

One exception to the configuration parameter behavior described above is the LOGFILSIZ database configuration parameter. Although this parameter is not replicated to the standby database, to guarantee identical log files on both databases, the standby database ignores the local LOGFILSIZ configuration and creates local log files that match the size of the log files on the primary database.

After a takeover, the original standby (new primary) will keep using the value that was set on the original primary until the database is restarted. At that point, the new primary will revert to the value configured locally. In addition, the new primary also truncates the current log file and resizes any pre-created log files.

If the databases keep switching roles via non-forced takeover and neither database is deactivated, then the log file size used will always be the one established by the very first primary. However, if there is a deactivate and then a restart on the original standby (new primary) then it would use the log file size configured locally. This log file size would continue to be used if the original primary takes over again. Only after a deactivate and restart on the original primary would the log file size revert back to the settings on the original primary.

Log receive buffer size on the standby database:

By default, the log receive buffer size on the standby database will be two times the value specified for the LOGBUFSZ configuration parameter on the primary database. There might be times when this size is not sufficient. For example, when

the HADR synchronization mode is asynchronous and the primary and standby databases are in peer state, if the primary database is experiencing a high transaction load, the log receive buffer on the standby database might fill to capacity and the log shipping operation from the primary database might stall. To manage these temporary peaks, you can increase the size of the log receive buffer on the standby database by modifying the DB2_HADR_BUF_SIZE registry variable.

Load operations and HADR:

If a load operation is executed on the primary database with the COPY YES option, the command will execute on the primary database and the data will be replicated to the standby database as long as the copy can be accessed through the path or device specified by the LOAD command. If the standby database cannot access the data, the table space in which the table is stored is marked bad on the standby database. The standby database will skip future log records that pertain to this table space. To ensure that the load operation can access the copy on the standby database, it is recommended that you use a shared location for the output file from the COPY YES option. Alternatively, you can deactivate the standby database while the load operation is performed, perform the load on the primary, place a copy of the output file in the standby path, and then activate the standby database.

If a load operation is executed on the primary database with the NONRECOVERABLE option, the command will execute on the primary database and the table on the standby database will be marked bad. The standby database will skip future log records that pertain to this table. You can choose to issue the LOAD command with the COPY YES and REPLACE options specified to bring the table back, or you can drop the table to recover the space.

Since executing a load operation with the COPY NO option is not supported with HADR, the command is automatically converted to a load operation with the NONRECOVERABLE option. To enable a load operation with the COPY NO option to be converted to a load operation with the COPY YES option, set the DB2_LOAD_COPY_NO_OVERRIDE registry variable on the primary database. This registry variable is ignored by the standby database. Ensure that the device or directory specified on the primary database can be accessed by the standby database using the same path, device, or load library.

If you are using Tivoli Storage Manager (TSM) to perform a load operation with the COPY YES option, you might need to set the VENDOROPT configuration parameter on the primary and standby databases. Depending on how TSM is configured, the values on the primary and standby databases might not be the same. Also, when using TSM to perform a load operation with the COPY YES option, you must issue the **db2adutl** command with the GRANT option to give the standby database read access for the files that are loaded.

If table data is replicated by a load operation with the COPY YES option specified, the indexes will be replicated as follows:

- If the indexing mode is set to REBUILD and the table attribute is set to LOG INDEX BUILD, or the table attribute is set to DEFAULT and the LOGINDEXBUILD database configuration parameter is set to ON, the primary database will include the rebuilt index object in the copy file to enable the standby database to replicate the index object. If the index object on the standby database is marked bad before the load operation, it will become usable again after the load operation as a result of the index rebuild.

- If the indexing mode is set to INCREMENTAL and the table attribute is set to LOG INDEX BUILD, or the table attribute is set to NULL and LOGINDEXBUILD database configuration parameter on the primary database is set to ON, the index object on the standby database is updated only if it is not marked bad before the load operation. Otherwise, the index is marked bad on the standby database.

HADR configuration parameters:

Several new database configuration parameters are available to support HADR. Setting these parameters does not change the role of a database. You must issue the START HADR or STOP HADR commands to change the role of a database.

HADR configuration parameters are not dynamic. Any changes made to an HADR configuration parameter are not effective until the database has been shut down and restarted. In a partitioned database environment, the HADR configuration parameters are visible and can be changed, but they are ignored.

The local host name of the primary database must be the same as the remote host name of the standby database, and the local host name of the standby database must be the same as the remote host name of the primary database. Use the HADR_LOCAL_HOST and HADR_REMOTE_HOST configuration parameters to set the local and remote hosts for each database. Configuration consistency for the local and remote host names is checked when a connection is established to ensure that the remote host specified is the expected database.

An HADR database can be configured to use either IPv4 or IPv6 to locate its partner database. If the host server does not support IPv6, the database will use IPv4. If the server does support IPv6, whether the database uses IPv4 or IPv6 depends upon the format of the address specified for the HADR_LOCAL_HOST and HADR_REMOTE_HOST configuration parameters. The database attempts to resolve the two parameters to the same IP format. The following table shows how the IP mode is determined for IPv6-enabled servers:

IP mode used for HADR_LOCAL_HOST	IP mode used for HADR_REMOTE_HOST	IP mode used for HADR communications
IPv4 address	IPv4 address	IPv4
IPv4 address	IPv6 address	Error
IPv4 address	hostname, maps to v4 only	IPv4
IPv4 address	hostname, maps to v6 only	Error
IPv4 address	hostname, maps to v4 and v6	IPv4
IPv6 address	IPv4 address	Error
IPv6 address	IPv6 address	IPv6
IPv6 address	hostname, maps to v4 only	Error
IPv6 address	hostname, maps to v6 only	IPv6
IPv6 address	hostname, maps to v4 and v6	IPv6
hostname, maps to v4 only	IPv4 address	IPv4
hostname, maps to v4 only	IPv6 address	Error
hostname, maps to v4 only	hostname, maps to v4 only	IPv4
hostname, maps to v4 only	hostname, maps to v6 only	Error
hostname, maps to v4 only	hostname, maps to v4 and v6	IPv4

IP mode used for HADR_LOCAL_HOST	IP mode used for HADR_REMOTE_HOST	IP mode used for HADR communications
hostname, maps to v6 only	IPv4 address	Error
hostname, maps to v6 only	IPv6 address	IPv6
hostname, maps to v6 only	hostname, maps to v4 only	Error
hostname, maps to v6 only	hostname, maps to v6 only	IPv6
hostname, maps to v6 only	hostname, maps to v4 and v6	IPv6
hostname, maps to v4 and v6	IPv4 address	IPv4
hostname, maps to v4 and v6	IPv6 address	IPv6
hostname, maps to v4 and v6	hostname, maps to v4 only	IPv4
hostname, maps to v4 and v6	hostname, maps to v6 only	IPv6
hostname, maps to v4 and v6	hostname, maps to v4 and v6	IPv6

The primary and standby databases can make HADR connections only if they use the same format. If one server is IPv6 enabled (but also supports IPv4) and the other server only supports IPv4, at least one of the HADR_LOCAL_HOST or HADR_REMOTE_HOST parameters must specify an IPv4 address. This tells the database to use IPv4 even if the server supports IPv6.

When you specify values for the high availability disaster recovery (HADR) local service and remote service parameters (HADR_LOCAL_SVC and HADR_REMOTE_SVC) while preparing an **update database configuration** command, the values you specify must be ports that are not in use for any other service, including other DB2 components or other HADR databases. In particular, you cannot set either parameter value to the TCP/IP port used by the server to await communications from remote clients (the SVCENAME database manager configuration parameter) or the next port (SVCENAME + 1).

If the primary and standby databases are on different machines, they can use the same port number or service name; otherwise, different values should be used. The HADR_LOCAL_SVC and HADR_REMOTE_SVC parameters can be set to either a port number or a service name.

The synchronization mode (HADR_SYNCMODE) and time-out period (HADR_TIMEOUT) must be identical on both the primary and standby databases. The consistency of these configuration parameters is checked when an HADR pair establishes a connection.

TCP connections are used for communication between the primary and standby databases. A primary database that is not connected to a standby database, either because it is starting up or because the connection is lost, will listen on its local port for new connections. A standby database that is not connected to a primary database will continue issuing connection requests to its remote host.

Although the local host and local service parameters (HADR_LOCAL_HOST, HADR_LOCAL_SVC) are only used on the primary database, you should still set them on the standby database to ensure that they are ready if the standby database has to take over as the primary database.

When the primary database starts, it waits for a standby to connect for a minimum of 30 seconds or for the number of seconds specified by the value of the HADR_TIMEOUT database configuration parameter, whichever is longer. If the

standby does not connect in the specified time, the startup will fail. (The one exception to this is when the START HADR command is issued with the BY FORCE option.)

After an HADR pair establishes a connection, they will exchange heart beat messages. The heartbeat interval is one-quarter of the value of the HADR_TIMEOUT database configuration parameter, or 30 seconds, whichever is shorter. The HADR_HEARTBEAT monitor element shows the number of heartbeats a database expected to receive but did not receive from the other database. If one database does not receive any message from the other database within the number of seconds specified by HADR_TIMEOUT, it will initiate a disconnect. This means that at most it takes the number of seconds specified by HADR_TIMEOUT for a primary to detect the failure of either the standby or the intervening network. In HADR peer state, problems with the standby or network the will only block primary transaction processing for the number of seconds specified by the HADR_TIMEOUT configuration parameter, at most. If you set this configuration parameter too low, you will receive false alarms and frequent disconnections.

Note: For maximal availability, as soon as the connection between the primary and the standby is closed (either because the standby closed the connection, a network error is detected, or timeout is reached), the primary drops out of peer state to avoid blocking transactions.

The following sample configuration is for the primary and standby databases.

On the primary:

```
HADR_LOCAL_HOST           host1.ibm.com
HADR_LOCAL_SVC            hadr_service
HADR_REMOTE_HOST          host2.ibm.com
HADR_REMOTE_SVC           hadr_service
HADR_REMOTE_INST          dbinst2
HADR_TIMEOUT              120
HADR_SYNCMODE             NEARSYNC
```

On the standby:

```
HADR_LOCAL_HOST           host2.ibm.com
HADR_LOCAL_SVC            hadr_service
HADR_REMOTE_HOST          host1.ibm.com
HADR_REMOTE_SVC           hadr_service
HADR_REMOTE_INST          dbinst1
HADR_TIMEOUT              120
HADR_SYNCMODE             NEARSYNC
```

Related concepts:

- "High availability disaster recovery overview" on page 221
- "Log archiving configuration for high availability disaster recovery (HADR)" on page 257

Related reference:

- "hadr_db_role - HADR database role configuration parameter" in *Performance Guide*
- "hadr_local_host - HADR local host name configuration parameter" in *Performance Guide*
- "hadr_local_svc - HADR local service name configuration parameter" in *Performance Guide*

- "hadr_remote_host - HADR remote host name configuration parameter" in *Performance Guide*
- "hadr_remote_inst - HADR instance name of the remote server configuration parameter" in *Performance Guide*
- "hadr_remote_svc - HADR remote service name configuration parameter" in *Performance Guide*
- "hadr_syncmode - HADR synchronization mode for log write in peer state configuration parameter" in *Performance Guide*
- "hadr_timeout - HADR timeout value configuration parameter" in *Performance Guide*
- "vendoropt - Vendor options configuration parameter" in *Performance Guide*

Database activation and deactivation in high availability disaster recovery (HADR)

If a standard database is started by a client connection, the database is shut down when the last client disconnects. If an HADR primary database is started by a client connection, it is equivalent to starting the database by using the ACTIVATE DATABASE command. To shut down an HADR primary database that was started by a client connection, you need to explicitly issue the DEACTIVATE DATABASE command.

On a standard database in rollforward pending state, the ACTIVATE DATABASE and DEACTIVATE DATABASE commands are not applicable. You can only continue rollforward, stop rollforward, or use START HADR start the database as an HADR standby database. Once a database is started as an HADR standby, you can use the ACTIVATE DATABASE and DEACTIVATE DATABASE commands to start and stop the database.

Activate a primary database using the following methods:
- client connection
- ACTIVATE DATABASE command
- START HADR command with the AS PRIMARY option

Deactivate a primary database using the following methods:
- DEACTIVATE DATABASE command
- db2stop command with the FORCE option

Activate a standby database using the following methods:
- ACTIVATE DATABASE command
- START HADR command with the AS STANDBY option

Deactivate a standby database using the following methods:
- DEACTIVATE DATABASE command
- db2stop command with the FORCE option

Related concepts:
- "Database configuration for high availability disaster recovery (HADR)" on page 249
- "High availability disaster recovery (HADR) commands overview" on page 234
- "High availability disaster recovery overview" on page 221

Related reference:

- "ACTIVATE DATABASE command" in *Command Reference*
- "db2stop - Stop DB2 command" in *Command Reference*
- "DEACTIVATE DATABASE command" in *Command Reference*

Automatic client reroute and high availability disaster recovery (HADR)

The automatic client reroute feature can be used with high availability disaster recovery to allow client applications to recover from a loss of communication with the server and to continue working with minimal interruption. Rerouting is only possible when an alternate database location has been specified at the server. Automatic client reroute is only supported with TCP/IP protocol.

Note: Client reroute is enabled by default if you set up HADR using the Set Up High Availability Disaster Recovery (HADR) Databases wizard in the Control Center.

You can use automatic client reroute with HADR to make client applications connect to the new primary database after a takeover operation. If automatic client reroute is not enabled, client applications will receive error message SQL30081, and no further attempts will be made to establish a connection with the server. The following example explains how to use the UPDATE ALTERNATE SERVER FOR DATABASE command to set up automatic client reroute with HADR.

Example

Your system is set up as follows:

- You have a client where database MUSIC is catalogued as being located at host HORNET.
- Database MUSIC is the primary database and its corresponding standby database, also MUSIC, resides on host MONTERO with port number 456, which is assigned by the SVCENAME configuration parameter.

To enable automatic client reroute, update the alternate server for database MUSIC on host HORNET:

```
db2 update alternate server for database music using hostname montero port 456
```

After this command is issued, the client must successfully connect to host HORNET to obtain the alternate server information. Then, if a communication error occurs between the client and database MUSIC at host HORNET, the client will first attempt to reconnect to database MUSIC at host HORNET. If this fails, the client will then attempt to establish a connection with the standby database MUSIC on host MONTERO.

Notes:

1. The alternate host location is stored in the system database directory file at the server.
2. To enable the automatic client reroute feature, you must use the UPDATE ALTERNATE SERVER FOR DATABASE command. Automatic client reroute does not use the HADR_REMOTE_HOST and HADR_REMOTE_SVC database configuration parameters.

Related concepts:

- "High availability disaster recovery (HADR) commands overview" on page 234

- "High availability disaster recovery overview" on page 221

Related tasks:

- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Related reference:

- "UPDATE ALTERNATE SERVER FOR DATABASE command" in *Command Reference*
- "db2UpdateAlternateServerForDB API - Update the alternate server for a database alias in the system database directory" in *Administrative API Reference*
- "Automatic client reroute roadmap" in *Administration Guide: Implementation*

Index logging and high availability disaster recovery (HADR)

Consider the following recommendations when setting configuration parameters for high availability disaster recovery (HADR) databases.

Using the LOGINDEXBUILD database configuration parameter

Recommendation: For HADR databases, set the LOGINDEXBUILD database configuration parameter to ON to ensure that complete information is logged for index creation, recreation, and reorganization. Although this means that index builds might take longer on the primary system and that more log space is required, the indexes will be rebuilt on the standby system during HADR log replay and will be available when a failover takes place. If index builds on the primary system are not logged and a failover occurs, any invalid indexes that remain after the failover is complete will have to be rebuilt before they can be accessed. While the indexes are being recreated, they cannot be accessed by any applications.

Note: If the LOG INDEX BUILD table attribute is set to its default value of NULL, DB2 will use the value specified for the LOGINDEXBUILD database configuration parameter. If the LOG INDEX BUILD table attribute is set to ON or OFF, the value specified for the LOGINDEXBUILD database configuration parameter will be ignored.

You might choose to set the LOG INDEX BUILD table attribute to OFF on one or more tables for either of the following reasons:

- You do not have enough active log space to support logging of the index builds.
- The index data is very large and the table is not accessed often; therefore, it is acceptable for the indexes to be recreated at the end of the takeover operation. In this case, set the INDEXREC configuration parameter to RESTART. Because the table is not frequently accessed, this setting will cause the system to recreate the indexes at the end of the takeover operation instead of waiting for the first time the table is accessed after the takeover operation.

If the LOG INDEX BUILD table attribute is set to OFF on one or more tables, any index build operation on those tables might cause the indexes to be recreated any time a takeover operation occurs. Similarly, if the LOG INDEX BUILD table attribute is set to its default value of NULL, and the LOGINDEXBUILD database configuration parameter is set to OFF, any index build operation on a table might

cause the indexes on that table to be recreated any time a takeover operation occurs. You can prevent the indexes from being recreated by taking one of the following actions:

- After all invalid indexes are recreated on the new primary database, take a backup of the database and apply it to the standby database. As a result of doing this, the standby database does not have to apply the logs used for recreating invalid indexes on the primary database, which would mark those indexes as rebuild required on the standby database.
- Set the LOG INDEX BUILD table attribute to ON, or set the LOG INDEX BUILD table attribute to NULL and the LOGINDEXBUILD configuration parameter to ON on the standby database to ensure that the index recreation will be logged.

Using the INDEXREC database configuration parameter

Recommendation: Set the INDEXREC database configuration parameter to RESTART (the default) on both the primary and standby databases. This will cause invalid indexes to be rebuilt after a takeover operation is complete. If any index builds have not been logged, this setting allows DB2 to check for invalid indexes and to rebuild them. This process takes place in the background, and the database will be accessible after the takeover operation has completed successfully.

If a transaction accesses a table that has invalid indexes before the indexes have been rebuilt by the background recreate index process, the invalid indexes will be rebuilt by the first transaction that accesses it.

Related reference:

- "indexrec - Index re-creation time configuration parameter" in *Performance Guide*
- "logindexbuild - Log index pages created configuration parameter" in *Performance Guide*
- "ALTER TABLE statement" in *SQL Reference, Volume 2*

Log archiving configuration for high availability disaster recovery (HADR)

The recommended approach to configuring log archiving for HADR is to configure both the primary and standby databases to have automatic log retrieval capability from all log archive locations used. Both the primary and standby databases need to be able to retrieve log files from all the log archive locations to which either of the databases might archive log files.

If either the standby database or the primary database is unable to access all log archive locations, then you must manually copy log files from the log archive to the following locations:

- the standby database logpath or archive location for local catch-up
- the primary database logpath or archive location for remote catch-up

Only the current primary database can perform log archiving. If the primary and standby databases are set up with separate archiving locations, logs are archived only to the primary database's archiving location. In the event of a takeover, the standby database becomes the new primary database and any logs archived from that point on are saved to the original standby database's archiving location. In such a configuration, logs are archived to one location or the other, but not both;

with the exception that following a takeover, the new primary database might archive a few logs that the original primary database had already archived.

After a takeover, if the new primary database (original standby database) experiences a media failure and needs to perform a restore and rollforward, it might need to access logs that only exist in the original primary database archive location.

The standby database will not delete a log file from its local logpath until it has been notified by the primary database that the primary database has archived it. This behavior provides added protection against the loss of log files. If the primary database fails and its log disk becomes corrupted before a particular log file is archived on the primary database, the standby database will not delete that log file from its own disk because it has not received notification that the primary database successfully archived the log file. If the standby database then takes over as the new primary database, it will archive that log file before recycling it. If both the *logarchmeth1* and *logarchmeth2* configuration parameters are in use, the standby database will not recycle a log file until the primary database has archived it using both methods.

Related concepts:

- "Log file management through log archiving" on page 49
- "Standby database states in high availability disaster recovery (HADR)" on page 226
- "Crash recovery" on page 10
- "Database configuration for high availability disaster recovery (HADR)" on page 249
- "High availability disaster recovery overview" on page 221

Related reference:

- "hadr_standby_log_file - HADR Standby Log File monitor element" in *System Monitor Guide and Reference*
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*
- "logarchmeth2 - Secondary log archive method configuration parameter" in *Performance Guide*
- "hadr_primary_log_file - HADR Primary Log File monitor element" in *System Monitor Guide and Reference*
- "Configuration parameters for database logging" on page 37

Cluster managers and high availability disaster recovery (HADR)

You can use high availability disaster recovery (HADR) with a cluster manager to enhance the availability of the DBMS. There are two ways you can configure HADR to do this:

- Set up an HADR pair where the primary and standby databases are serviced by the same cluster manager.

  This configuration is best suited to environments where the primary and standby databases are located at the same site and where the fastest possible failover is required. These environments would benefit from using HADR to maintain DBMS availability, rather using crash recovery or another recovery method.

You can use the cluster manager to quickly detect a problem and to initiate a takeover operation. Because HADR requires separate storage for the DBMS, the cluster manager should be configured with separate volume control. This configuration prevents the cluster manager from waiting for failover to occur on the volume before using the DBMS on the standby system. You can use the automatic client reroute feature to redirect client applications to the new primary database.

- Set up an HADR pair where the primary and standby databases are not serviced by the same cluster manager.

  This configuration is best suited to environments where the primary and standby databases are located at different sites and where high availability is required for disaster recovery in the event of a complete site failure. There are several ways you can implement this configuration. When an HADR primary or standby database is part of a cluster, there are two possible failover scenarios.

  – If a partial site failure occurs and a node to which the DBMS can fail over remains available, you can choose to perform a cluster failover. In this case, the IP address and volume failover is performed using the cluster manager; HADR is not affected.
  – If a complete site failure occurs where the primary database is located, you can use HADR to maintain DBMS availability by initiating a takeover operation. If a complete site failure occurs where the standby database is located, you can repair the site or move the standby database to another site.

Related concepts:

- "Automatic client reroute and high availability disaster recovery (HADR)" on page 255
- "High availability disaster recovery overview" on page 221

Related tasks:

- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Switching database roles in high availability disaster recovery (HADR)

During high availability disaster recovery (HADR), use the TAKEOVER HADR command to switch the roles of the primary and standby databases.

Restrictions:

- The TAKEOVER HADR command can only be issued on the standby database. If the primary database is not connected to the standby database when the command is issued, the takeover operation will fail.
- The TAKEOVER HADR command can only be used to switch the roles of the primary and standby databases if the databases are in peer state. If the standby database is in any other state, an error message will be returned.

Procedure:

You can switch the HADR database roles using the command line processor (CLP), the Manage High Availability Disaster Recovery (HADR) window in the Control Center, or the **db2HADRTakeover** application programming interface (API).

To use the CLP to initiate a takeover operation on the standby database, issue the TAKEOVER HADR command without the BY FORCE option on the standby database.

In the following example, the takeover operation takes place on the standby database LEAFS:

```
TAKEOVER HADR ON DB LEAFS
```

A log full error is slightly more likely to occur immediately following a takeover operation. To limit the possibility of such an error, an asynchronous buffer pool flush is automatically started at the end of each takeover. The likelihood of a log full error decreases as the asynchronous buffer pool flush progresses. Additionally, if your configuration provides a sufficient amount of active log space, a log full error is even more unlikely. If a log full error does occur, the current transaction is aborted and rolled back.

Usage note: Issuing the TAKEOVER HADR command without the BY FORCE option will cause any applications currently connected to the HADR primary database to be forced off. This action is designed to work in coordination with automatic client reroute to assist in rerouting clients to the new HADR primary database after a role switch. However, if the forcing off of applications from the primary would be disruptive in your environment, you might want to implement your own procedure to shut down such applications prior to performing a role switch, and then restart them with the new HADR primary database as their target after the role switch is completed.

To open the Takeover HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Ensure that the databases are in peer state
3. Click Takeover HADR. The Takeover HADR window opens.
4. Select that you want to switch the database roles.
5. If both databases in the HADR pair have been started as standby databases, select one of the databases to take over as the primary database.
6. Click OK. The window closes. A progress indicator might open to indicate when the command is running. When it completes, you will get a notification indicating whether or not it is successful.
7. Refresh the Manage High Availability Disaster Recovery window to ensure that the databases have switched roles.
8. If you are not using the automatic client reroute feature, redirect client applications to the new primary database.

Additional information is provided through the contextual help facility within the Control Center.

Related concepts:

- "Automatic client reroute and high availability disaster recovery (HADR)" on page 255
- "High availability disaster recovery overview" on page 221

- "Standby database states in high availability disaster recovery (HADR)" on page 226

Related tasks:

- "Reintegrating a database after a takeover operation" on page 268

Performing an HADR failover operation

When you want the current standby database to become the new primary database because the current primary database is not available, you can perform a failover.

Warning: This procedure might cause a loss of data. Review the following information before performing this emergency procedure:

- Ensure that the primary database is no longer processing database transactions. If the primary database is still running, but cannot communicate with the standby database, executing a forced takeover operation (issuing the TAKEOVER HADR command with the BY FORCE option) could result in two primary databases. When there are two primary databases, each database will have different data, and the two databases can no longer be automatically synchronized.
  - Deactivate the primary database or stop its instance, if possible. (This might not be possible if the primary system has hung, crashed, or is otherwise inaccessible.) After a takeover operation is performed, if the failed database is later restarted, it will not automatically assume the role of primary database.
- The likelihood and extent of transaction loss depends on your specific configuration and circumstances:
  - If the primary database fails while in peer state and the synchronization mode is synchronous (SYNC), the standby database will not lose transactions that were reported committed to an application before the primary database failed.
  - If the primary database fails while in peer state and the synchronization mode is near synchronous (NEARSYNC), the standby database can only lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
  - If the primary database fails while in peer state and the synchronization mode is asynchronous (ASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
  - If the primary database fails while in remote catchup pending state, transactions that have not been received and processed by the standby database will be lost.

    Note: Any log gap shown in the database snapshot will represent the gap at the last time the primary and standby databases were communicating with each other; the primary database might have processed a very large number of transactions since that time.

- Ensure that any application that connects to the new primary (or that is rerouted to the new primary by client reroute), is prepared to handle the following:
  - There is data loss during failover. The new primary does not have all of the transactions committed on the old primary. This can happen even when the HADR_SYNCMODE configuration parameter is set to SYNC. Because an HADR standby applies logs sequentially, you can assume that if a transaction in an SQL session is committed on the new primary, all previous transactions in the same session have also been committed on the new primary. The commit sequence of transactions across multiple sessions can be determined only with detailed analysis of the log stream.
  - It is possible that a transaction can be issued to the original primary, committed on the original primary and replicated to the new primary (original standby), but not be reported as committed because the original primary crashed before it could report to the client that the transaction was committed. Any application you write should be able to handle that transactions issued to the original primary, but not reported as committed on the original primary, are committed on the new primary (original standby).
  - Some operations are not replicated, such as changes to database configuration and to external UDF objects.

Restrictions:

- The TAKEOVER HADR command can only be issued on the standby database.
- HADR does not interface with the DB2 fault monitor (db2fm) which can be used to automatically restart a failed database. If the fault monitor is enabled, you should be aware of possible fault monitor action on a presumably failed primary database.
- A takeover operation can only take place if the primary and standby databases are in peer state or the standby database is in remote catchup pending state. If the standby database is in any other state, an error will be returned.

  Note: You can make a standby database that is in local catchup state available for normal use by converting it to a standard database. To do this, shut the database down by issuing the DEACTIVATE DATABASE command, and then issue the STOP HADR command. Once HADR has been stopped, you must complete a rollforward operation on the former standby database before it can be used. A database cannot rejoin an HADR pair after it has been converted from a standby database to a standard database. To restart HADR on the two servers, follow the procedure for initializing HADR.

Procedure:

In a failover scenario, a takeover operation can be performed through the command line processor (CLP), the Manage High Availability Disaster Recovery window in the Control Center, or the **db2HADRTakeover** application programming interface (API).

The following procedure shows you how to initiate a failover on the primary or standby database using the CLP:

1. Completely disable the failed primary database. When a database encounters internal errors, normal shutdown commands might not completely shut it

down. You might need to use operating system commands to remove resources such as processes, shared memory, or network connections.

2. Issue the TAKEOVER HADR command with the BY FORCE option on the standby database. In the following example the failover takes place on database LEAFS:

   ```
   TAKEOVER HADR ON DB LEAFS BY FORCE
   ```

   The BY FORCE option is required because the primary is expected to be offline.

   If the primary database is not completely disabled, the standby database will still have a connection to the primary and will send a message to the primary database asking it to shutdown. The standby database will still switch to the role of primary database whether or not it receives confirmation from that the primary database has been shutdown.

To open the Takeover HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Click Takeover HADR. The Takeover HADR window opens.
3. Select that you want to execute a failover operation.
4. If both databases in the HADR pair have been started as standby databases, select one of the databases to take over as the primary database.
5. Click OK. The window closes. A progress indicator might open to indicate when the command is running. When it completes, you will get a notification indicating whether or not it is successful.
6. Refresh the Manage High Availability Disaster Recovery window to ensure that the standby database has taken over as the new primary.
7. If you are not using the automatic client reroute feature, redirect client applications to the new primary database.

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

Related concepts:

- "High availability disaster recovery overview" on page 221
- "Standby database states in high availability disaster recovery (HADR)" on page 226
- "Synchronization modes for high availability disaster recovery (HADR)" on page 229

Related tasks:

- "Initializing high availability disaster recovery (HADR)" on page 238
- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Related reference:

- "DEACTIVATE DATABASE command" in *Command Reference*
- "ROLLFORWARD DATABASE " on page 168
- "STOP HADR" on page 246
- "TAKEOVER HADR" on page 264

TAKEOVER HADR

Instructs an HADR standby database to take over as the new HADR primary database for the HADR pair.

Authorization:

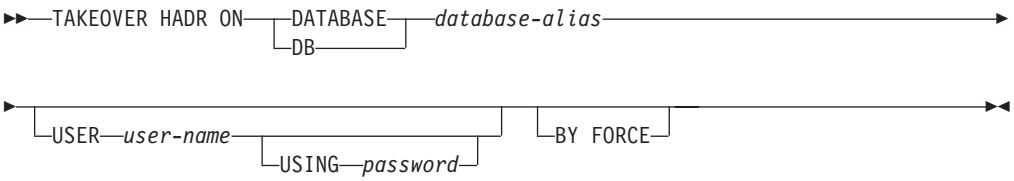
One of the following:

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

Required connection:

Instance. The command establishes a database connection if one does not exist, and closes the database connection when the command completes.

Command syntax:



Command parameters:

DATABASE *database-alias*
Identifies the current HADR standby database that should take over as the HADR primary database.

USER *user-name*
Identifies the user name under which the takeover operation is to be started.

USING *password*
The password used to authenticate *user-name*.

BY FORCE
Specifies that the database will not wait for confirmation that the original HADR primary database has been shut down. This option is required if the HADR pair is not in peer state.

Usage notes:

The following table shows the behavior of the **TAKEOVER HADR** command when issued on an active standby for each possible state and option combination. An error message is returned if this command is issued on an inactive standby database.

Standby state	BY FORCE option used	Takeover behavior
Local catchup or remote catchup	No	Error message returned

Standby state	BY FORCE option used	Takeover behavior
Local catchup or remote catchup	Yes	Error message returned
Peer	No	Primary database and standby database switch roles. If no failure is encountered during takeover, there will be no data loss. However, if failures are encountered during takeover, data loss might occur and the roles of the primary and standby might or might not have been changed. The following is a guideline for handling failures during a takeover in which the primary and standby switch roles: 1. If a failure occurs during a takeover operation, the roles of the HADR databases might or might not have been changed. If possible, make sure both databases are online. Check the HADR role of the available database or databases using the Snapshot Monitor, or by checking the value of the database configuration parameter *hadr_db_role*. 2. If the intended new primary is still in standby role, and takeover is still desired, re-issue the **TAKEOVER HADR** command (see the next guideline regarding the `BY FORCE` option). 3. It is possible to end up with both databases in standby role. In that case, the **TAKEOVER HADR** command with the `BY FORCE` option can be issued at whichever node should now become the primary. The `BY FORCE` option is required in this case because the two standbys cannot establish the usual HADR primary-standby connection.
Peer	Yes	The standby notifies the primary to shut itself (the primary) down. The standby stops receiving logs from the primary, finishes replaying the logs it has already received, and then becomes a primary. The standby does *not* wait for any acknowledgement from the primary to confirm that it has received the takeover notification or that it has shut down. Because of this, if the primary is processing transactions at the time of the takeover, it is unlikely that it will be able to be later restarted as a standby. It is recommended that you shut down the primary database first before issuing a **TAKEOVER HADR** command with the `BY FORCE` option.
Remote catchup pending	No	Error message returned.
Remote catchup pending	Yes	The standby becomes a primary.

When issuing the **TAKEOVER HADR** command, the corresponding error codes might be generated: SQL01767N, SQL01769N, or SQL01770N with a reason code of 98. The reason code indicates that there is no installed license for HADR on the server where the command was issued. To correct the problem, install a valid HADR license using the **db2licm** or install a version of the server that contains a valid HADR license as part of its distribution.

Related tasks:

- "Switching database roles in high availability disaster recovery (HADR)" on page 259

db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database

Instructs a standby database to take over as the primary database. This API can be called against a standby database only.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. The API establishes a database connection if one does not exist, and closes the database connection when the API completes.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HADRTakeover (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRTakeoverStruct
{
   char *piDbAlias;
   char *piUserName;
   char *piPassword;
   db2Uint16 iByForce;
} db2HADRTakeoverStruct;

SQL_API_RC SQL_API_FN
  db2gHADRTakeover (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRTakeoverStruct
{
   char *piDbAlias;
   db2Uint32 iAliasLen;
   char *piUserName;
   db2Uint32 iUserNameLen;
   char *piPassword;
   db2Uint32 iPasswordLen;
   db2Uint16 iByForce;
} db2gHADRTakeoverStruct;
```

db2HADRTakeover API parameters:

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

pParmStruct
Input. A pointer to the db2HADRTakeoverStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2HADRTakeoverStruct data structure parameters:

piDbAlias
Input. A pointer to the database alias.

piUserName
Input. A pointer to the user name under which the command will be executed.

piPassword
Input. A pointer to a string containing the password.

iByForce
Input. Valid values are:

DB2HADR_NO_FORCE
Specifies that a takeover occurs only if the two systems are in peer state with communication established; this results in a role reversal between the HADR primary and HADR standby databases.

DB2HADR_FORCE
Specifies that the standby database takes over as the primary database without waiting for confirmation that the original primary database has been shut down. Forced takeover must be issued when the standby database is in either remote catchup pending or peer state.

db2gHADRTakeoverStruct data structure specific parameters:

iAliasLen
Input. Specifies the length in bytes of the database alias.

iUserNameLen
Input. Specifies the length in bytes of the user name.

iPasswordLen
Input. Specifies the length in bytes of the password.

Related tasks:

- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "db2HADRStart - Start high availability disaster recovery (HADR) operations" on page 242
- "db2HADRStop - Stop high availability disaster recovery (HADR) operations" on page 247

Reintegrating a database after a takeover operation

If you executed a takeover operation in a high availability disaster recovery (HADR) environment because the primary database failed, you can bring the failed database back online and use it as a standby database or return it to its status as primary database.

To reintegrate the failed primary database into the HADR pair as the new standby database:

1. Repair the system where the original primary database resided. This could involve repairing failed hardware or rebooting the crashed operating system.
2. Restart the failed primary database as a standby database. In the following example, database LEAFS is started as a standby database:

   ```
   START HADR ON DB LEAFS AS STANDBY
   ```

   Note: Reintegration will fail if the two copies of the database have incompatible log streams. In particular, HADR requires that the original primary database did not apply any logged operation that was never reflected on the original standby database before it took over as the new primary database. If this did occur, you can restart the original primary database as a standby database by restoring a backup image of the new primary database or by initializing a split mirror.

   Successful return of this command does not indicate that reintegration has succeeded; it means only that the database has been started. Reintegration is still in progress. If reintegration subsequently fails, the database will shut itself down. You should monitor standby states using the GET SNAPSHOT FOR DATABASE command or the db2pd tool to make sure that the standby database stays online and proceeds with the normal state transition. If necessary, you can check the administration log file db2diag.log to find out the status of the database.

After the original primary database has rejoined the HADR pair as the standby database, you can choose to perform a failback operation to switch the roles of the databases to enable the original primary database to be once again the primary database. To perform this failback operation, issue the following command on the standby database:

```
TAKEOVER HADR ON DB LEAFS
```

Notes:

1. If the HADR databases are not in peer state or the pair is not connected, this command will fail.
2. Open sessions on the primary database are forced closed and inflight transactions are rolled back.
3. When switching the roles of the primary and standby databases, the BY FORCE option of the TAKEOVER HADR command cannot be specified.

Related concepts:

- "Standby database states in high availability disaster recovery (HADR)" on page 226
- "Synchronization modes for high availability disaster recovery (HADR)" on page 229

Related tasks:

- "Switching database roles in high availability disaster recovery (HADR)" on page 259

Related reference:

- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*
- "TAKEOVER HADR" on page 264

Performing a rolling upgrade in a high availability disaster recovery environment

Use this procedure in a high availability disaster recovery (HADR) environment when you upgrade software (operating system or DB2 database system) or hardware, or when you make changes to database configuration parameters. This procedure keeps database service available throughout the upgrade process, with only a momentary service interruption when processing is switched from one database to the other. Because HADR performs optimally when both the primary and standby databases are on identical systems, you should apply changes to both systems as quickly as possible.

Note: All DB2 database system fix packs and upgrades should be implemented in a test environment before being applied to your production system.

Prerequisites:

The HADR pair should be in peer state before starting the rolling upgrade.

Restrictions:

This procedure will not work to migrate from an earlier to a later version of a DB2 database system; for example, you cannot use this procedure to migrate from a version 8 to a version 9 database system. You can use this procedure to update your database system from one modification level to another only, for example by applying a fix pack.

This procedure will not work if you update the DB2 HADR configuration parameters. Updates to HADR configuration parameters should be made separately. Because HADR requires the parameters on the primary and standby to be the same, this might require both the primary and standby databases to be deactivated and updated at the same time.

Procedure:

To perform a rolling upgrade in an HADR environment:

1. Upgrade the system where the standby database resides:
   a. Use the DEACTIVATE DATABASE command to shut down the standby database.
   b. If necessary, shut down the instance on the standby database.
   c. Make changes to one or more of the following: the software, the hardware, or the DB2 configuration parameters.

      Note: You cannot change any HADR configuration parameters when performing a rolling upgrade.

d. If necessary, restart the instance on the standby database.
e. Use the ACTIVATE DATABASE command to restart the standby database.
f. Ensure that the standby database enters peer state. Use the GET SNAPSHOT command to check this.

2. Switch the roles of the primary and standby databases:
   a. Issue the TAKEOVER HADR command on the standby database.
   b. Direct clients to the new primary database. This can be done using automatic client reroute.

   Note: Because the standby database takes over as the primary database, the new primary database is now upgraded. If you are applying a DB2 database system fix pack, the TAKEOVER HADR command will change the role of the original primary database to standby database. However, the command will not let the new standby database connect to the newly upgraded primary database. Because the new standby database uses an older version of the DB2 database system, it might not understand the new log records generated by the upgraded primary database, and it will be shut down. In order for the new standby database to reconnect with the new primary database (that is, for the HADR pair to reform), the new standby database must also be upgraded.
3. Upgrade original primary database (which is now the standby database) using the same procedure as in Step 1 above. When you have done this, both databases are upgraded and connected to each other in HADR peer state. The HADR system provides full database service and full high availability protection.
4. Optional. To return to your original configuration, switch the roles of the primary and standby database as in step 2.

Related concepts:

- "Automatic client reroute and high availability disaster recovery (HADR)" on page 255
- "Standby database states in high availability disaster recovery (HADR)" on page 226
- "System requirements for high availability disaster recovery (HADR)" on page 222

Related reference:

- "ACTIVATE DATABASE command" in *Command Reference*
- "DEACTIVATE DATABASE command" in *Command Reference*
- "GET SNAPSHOT command" in *Command Reference*

Monitoring high availability disaster recovery (HADR)

You can use the following methods to monitor the status of your HADR databases.

db2pd utility
This utility retrieves information from the DB2 memory sets. For example, to view information about high availability disaster recovery for database MYDB, issue the following:

```
db2pd -db mydb -hadr
```

GET SNAPSHOT FOR DATABASE command
This command collects status information and formats the output. The

information returned represents a snapshot of the database manager operational status at the time the command was issued. HADR information appears in the output under the heading *HADR status*.

db2GetSnapshot API
This API collects database manager monitor information and returns it to a user-allocated data buffer. The information returned represents a snapshot of the database manager operational status at the time the API was called.

HADR configuration parameters are not dynamic:

If you change a parameter while the HADR database is online, the changes are visible when you issue a **db2 get db cfg** for the database. However, the changes are not effective until you stop and restart the database. To retrieve the parameters that are currently effective, use the GET SNAPSHOT command, the db2pd tool, or the snapshot monitor API.

HADR database roles:

The current role of a database is indicated by the database configuration parameter *hadr_db_role*. Valid values for this configuration parameter are PRIMARY, STANDBY, or STANDARD (the latter indicates the database is not an HADR database).

Status of the standby database:

When a database is in the standby role, it is also in rollforward pending state. Consequently, the standby database configuration will indicate:

```
Rollforward pending                                    = DATABASE
Restore pending                                        = YES
```

Related reference:

- "db2GetSnapshot API - Get a snapshot of the database manager operational status" in *Administrative API Reference*
- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*
- "GET SNAPSHOT command" in *Command Reference*
- "High availability disaster recovery monitor elements" in *System Monitor Guide and Reference*

High availability disaster recovery (HADR) performance

For optimum HADR performance, consider the following recommendations for managing your system:

- Network bandwidth must be greater than the database log generation rate.
- Network delays affect the primary only in SYNC and NEARSYNC modes.
- The slowdown in system performance as a result of using SYNC mode can be significantly larger than that of the other synchronization modes. In SYNC mode, the primary database sends log pages to the standby database only after the log pages have been successfully written to the primary database log disk. In order to protect the integrity of the system, the primary database waits for an acknowledgement from the standby before notifying an application that a transaction was prepared or committed. The standby database sends the

acknowledgement only after it writes the received log pages to the standby database disk. The resulting overhead is: the log write on the standby database plus round-trip messaging.

- In NEARSYNC mode, the primary database writes and sends log pages in parallel. The primary then waits for an acknowledgement from the standby. The standby database acknowledges as soon as the log pages are received into its memory. On a fast network, the overhead to the primary database is minimal. The acknowledgement might have already arrived by the time the primary database finishes local log write.
- For ASYNC mode, the log write and send are also in parallel; however, in this mode the primary database does not wait for an acknowledgement from the standby. Therefore, network delay is not an issue. Performance overhead is even smaller with ASYNC mode than with NEARSYNC mode.
- For each log write on the primary, the same log pages are also sent to the standby. Each write operation is called a *flush*. The size of the flush is limited to the log buffer size on the primary database (which is controlled by the database configuration parameter LOGBUFSZ). The exact size of each flush is nondeterministic. A larger log buffer does not necessarily lead to a larger flush size.
- The standby database should be powerful enough to replay the logged operations of the database as fast as they are generated on the primary. Identical primary and standby hardware is recommended.
- In most systems, the logging capability is not driven to its limit. Even in SYNC mode, there might not be an observable slow down on the primary database. For example, if the limit of logging is 40 Mb per second with HADR enabled, but the system was just running at 30 Mb per second before HADR is enabled, then you might not notice any difference in overall system performance.

Network congestion:

If the standby database is too slow replaying log pages, its log-receiving buffer might fill up, thereby preventing the buffer from receiving more log pages. In SYNC and NEARSYNC modes, if the primary database flushes its log buffer one more time, the data will likely be buffered in the network pipeline consisting of the primary machine, the network, and the standby database. Because the standby database does not have free buffer to receive the data, it cannot acknowledge, so the primary database becomes blocked while waiting for the standby database's acknowledgement.

In ASYNC mode, the primary database continues to send log pages until the pipeline fills up and it cannot send additional log pages. This condition is called *congestion*. Congestion is reported by the hadr_connect_status monitor element. For SYNC and NEARSYNC modes, the pipeline can usually absorb a single flush and congestion will not occur. However, the primary database remains blocked waiting for an acknowledgement from the standby database on the flush operation.

Congestion can also occur if the standby database is replaying log records that take a long time to replay, such as database or table reorganization log records.

Increasing the size of the standby database log-receiving buffer can help to reduce congestion, although it might not remove all of the causes of congestion. By default, the size of the standby database log-receiving buffer is two times the size of the primary database log-writing buffer. The database configuration parameter

LOGBUFSZ specifies the size of the primary database log-writing buffer. The DB2 registry variable DB2_HADR_BUF_SIZE can be used to tune the size of the standby database log-receiving buffer.

Related concepts:

- "Synchronization modes for high availability disaster recovery (HADR)" on page 229

db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database

Chapter 8. Cluster support on AIX

High Availability Cluster Multi-Processing support

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 can 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 partitioned database environment, DB2 database runs in mutual takeover mode with database partitions on each node. One exception is a scenario in which the catalog partition 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 might 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 database 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 database 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 3. 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 is packaged with DB2 Enterprise Server Edition to assist in configuring for HACMP ES failover or recovery in either hot standby or mutual takeover nodes. The script file is called `rc.db2pe.ee` for a single node and `rc.db2pe.eee` for multiple nodes. They are located in the `sqllib/samples/hacmp/es` directory. Copy the appropriate file to `/usr/bin` and rename it to `rc.db2pe`.

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 can 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 can 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 database environments on AIX see the following white papers which are available from the ″DB2 Database for Linux, UNIX,and Windows 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″
- ″Automating IBM DB2 UDB HADR with HACMP″

Related reference:

- “db2start - Start DB2 command” in *Command Reference*

Chapter 9. Cluster support on the Windows operating system

Microsoft Cluster Server support

Introduction

Microsoft Cluster Server (MSCS) is a feature of Windows 2000 Server and Windows Server 2003 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 Workgroup Server Edition
- DB2 Enterprise Server Edition (DB2 ESE)
- DB2 Connect Enterprise Server 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.

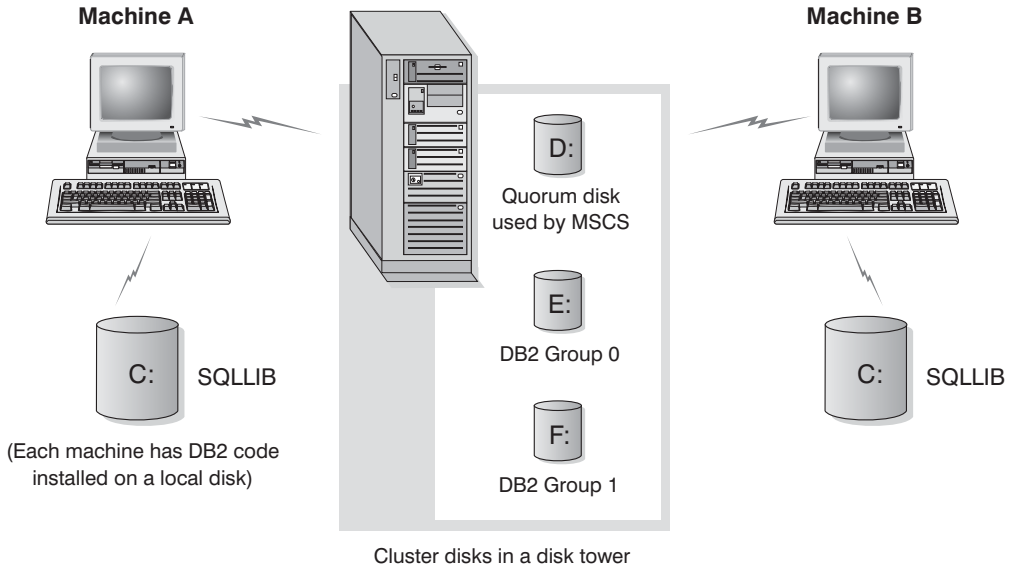


Figure 21. Example MSCS Configuration

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 (for example tables, indexes, log files) 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 DB2 Server. Each DB2 Server resource manages a DB2 instance, and when running in a partitioned database environment, each DB2 Server resource manages a database partition. The name of the DB2 Server resource is the instance name, although in the case of a partitioned database environment, the name of the DB2 Server resource consists of both the instance name and the database 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 might 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 can 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 environment, you are running multiple logical nodes on a machine and it fails, the logical nodes will be started on the failover machine. Figure 22 shows an example of a hot standby configuration.

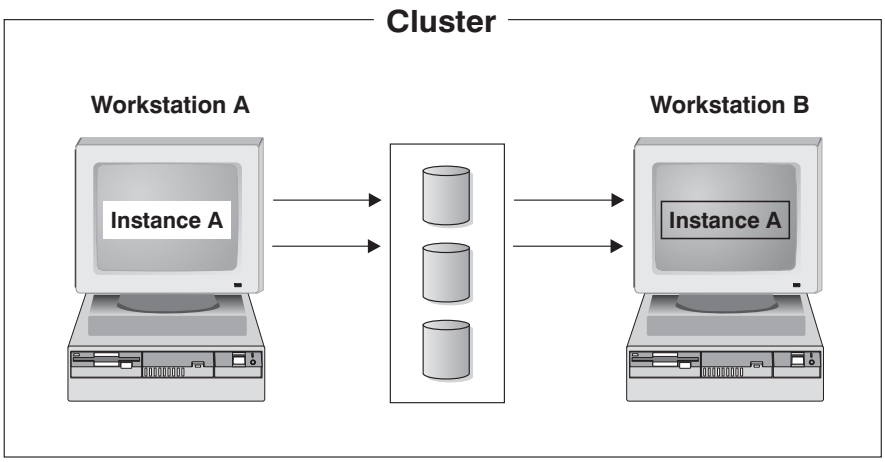


Figure 22. Hot Standby Configuration

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 23 on page 284 shows an example of a mutual takeover configuration.

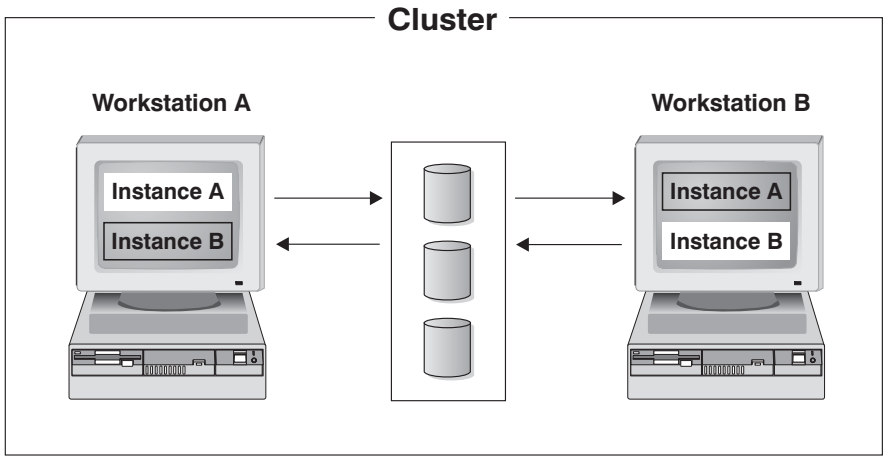


Figure 23. Mutual Takeover Configuration

For detailed information on the implementation and design of highly available IBM DB2 database environments on the Windows Operating System see the following white papers which are available from the ″DB2 Database for Linux, UNIX,and Windows 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″

Related concepts:

- “High availability” on page 207

Chapter 10. Cluster support for the Solaris Operating Environment

Cluster support for the Solaris operating system

High availability in the Solaris operating system can be achieved through DB2 working with Sun Cluster, or Veritas Cluster Server (VCS). For information about Sun Cluster, see the white paper entitled "DB2 Universal Database and High Availability on Sun Cluster 3.X", which is available from the "DB2 Database for Linux, UNIX, and Windows 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 available from the "IBM Support and downloads" Web site (http://www.ibm.com/support/docview.wss?uid=swg21045033).

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 24 on page 286.

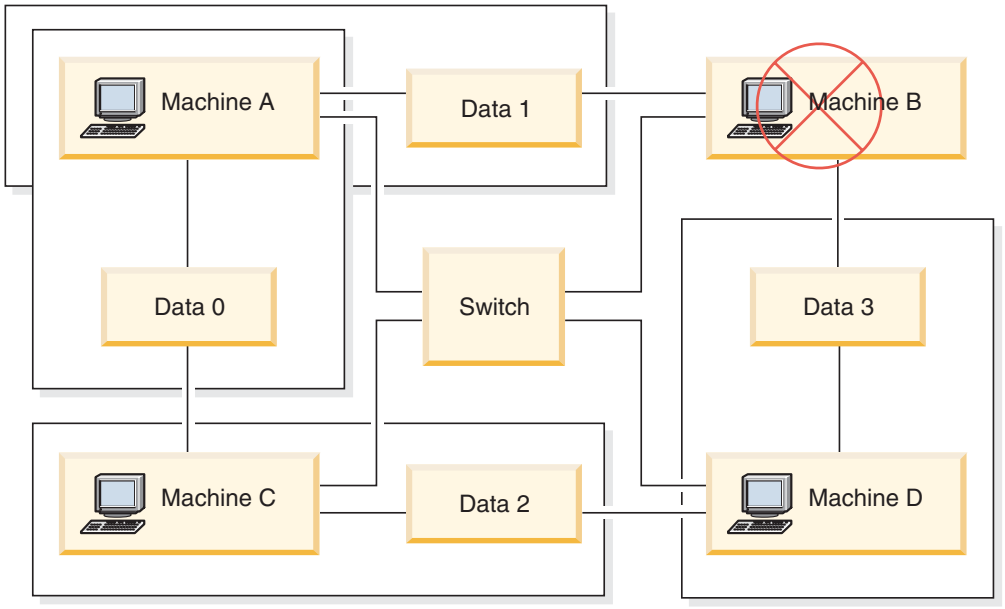


Figure 24. Failover. When Machine B fails its data service is moved to another machine in the cluster so that the data can still be accessed.

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 failover of the 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:

- "Sun Cluster 3.0 support" on page 287
- "VERITAS Cluster Server support" on page 290

Sun Cluster 3.0 support

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 25 on page 288).

Figure 25. DB2 database, Sun Cluster 3.0, and High Availability. The relationship between DB2 database, 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 might experience a brief interruption in service and might 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 device's 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 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 Database for Lunix, UNIX, and Windows Support″ web site (http://www.ibm.com/software/data/pubs/papers/).

Related concepts:

- "Cluster support for the Solaris operating system" on page 285
- "VERITAS Cluster Server support" on page 290

VERITAS Cluster Server support

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 binaries and shared disks between nodes for the DB2 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 database, 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 database service), either VERITAS Cluster Server or the VCS HA-DB2 Agent, or both will detect the failure and automatically take steps to restore the service. The steps take to restore the service can include restarting the DB2 database system on the same node or moving DB2 database system 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 (that is, 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 might or might 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 database), users might see a brief interruption in service and might need to reconnect after the failover has completed. In application environments with stateless connections (like NFS), users might 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 (for maintenance and upgrades). Failovers can also be initiated manually. If a hardware or operating system upgrade must be performed on a particular node, the DB2 database system can be migrated to another node in the cluster, the upgrade can be performed, and then the DB2 database system 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 database system is a crash tolerant application.

For information on how to decrease the amount of time it takes to perform a failover using a VERITAS CFS, CVM, and VCS solution, see the white paper entitled “DB2 UDB Version 8 and VERITAS Database Edition: Accelerating Failover Times in DB2 UDB Database Environments”, which is available from the “DB2

Database for Linux, UNIX, and Windows Support" web site (http://www.ibm.com/software/data/pubs/papers/).

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, 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 (that is, 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 the DB2 database application. 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 (that is, 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 database configuration, regardless of whether it is partitioned database environment 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 might be hosting several highly available applications there might 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 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:

- “Cluster support for the Solaris operating system” on page 285
- “Sun Cluster 3.0 support” on page 287

Part 3. Appendixes

Appendix A. How to read the syntax diagrams

Throughout this book, syntax is described using the structure defined as follows:

Read the syntax diagrams from left to right and top to bottom, following the path of the line.

The ►►— symbol indicates the beginning of a syntax diagram.

The —► symbol indicates that the syntax is continued on the next line.

The ►— symbol indicates that the syntax is continued from the previous line.

The —►◄ symbol indicates the end of a syntax diagram.

Syntax fragments start with the ├— symbol and end with the —┤ symbol.

Required items appear on the horizontal line (the main path).

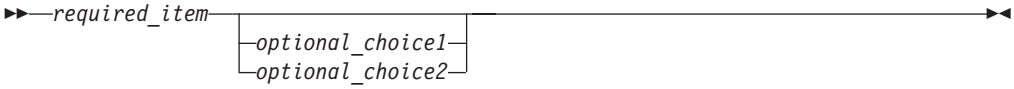
Optional items appear below the main path.

If an optional item appears above the main path, that item has no effect on execution, and is used only for readability.

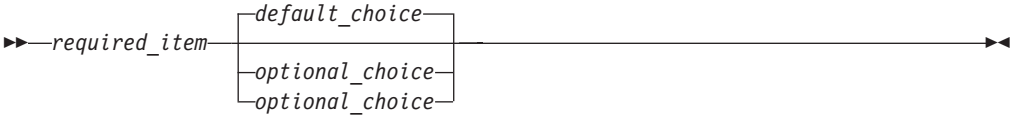
If you can choose from two or more items, they appear in a stack.

If you *must* choose one of the items, one item of the stack appears on the main path.

If choosing one of the items is optional, the entire stack appears below the main path.



If one of the items is the default, it will appear above the main path, and the remaining choices will be shown below.



An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.

If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

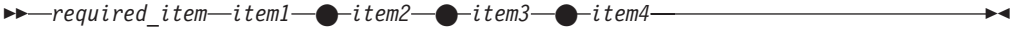
Keywords appear in uppercase (for example, FROM). They must be spelled exactly as shown. Variables appear in lowercase (for example, column-name). They represent user-supplied names or values in the syntax.

If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sometimes a single variable represents a larger fragment of the syntax. For example, in the following diagram, the variable parameter-block represents the whole syntax fragment that is labeled **parameter-block**:

parameter-block:

Adjacent segments occurring between "large bullets" (●) may be specified in any sequence.



The above diagram shows that item2 and item3 may be specified in either order. Both of the following are valid:

```
required_item item1 item2 item3 item4
required_item item1 item3 item2 item4
```

How to read the syntax diagrams

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 or XQuery statement. Some SQLCODE values have associated SQLSTATE values.

You can use the information contained in this topic 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).

Related concepts:

- "Introduction to Messages" in *Message Reference Volume 1*

Appendix C. Additional DB2 commands

This appendix describes recovery-related system and CLP commands that are not discussed in detail in this manual.

System commands

db2adutl - Managing DB2 objects within TSM

Allows users to query, extract, verify, and delete backup images, logs, and load copy images saved using Tivoli Storage Manager (TSM). Also allows users to grant and revoke access to objects on a TSM server.

On UNIX operating systems, this utility is located in the `sqllib/adsm` directory. On Windows operating systems, it is located in `sqllib\bin`.

Authorization:

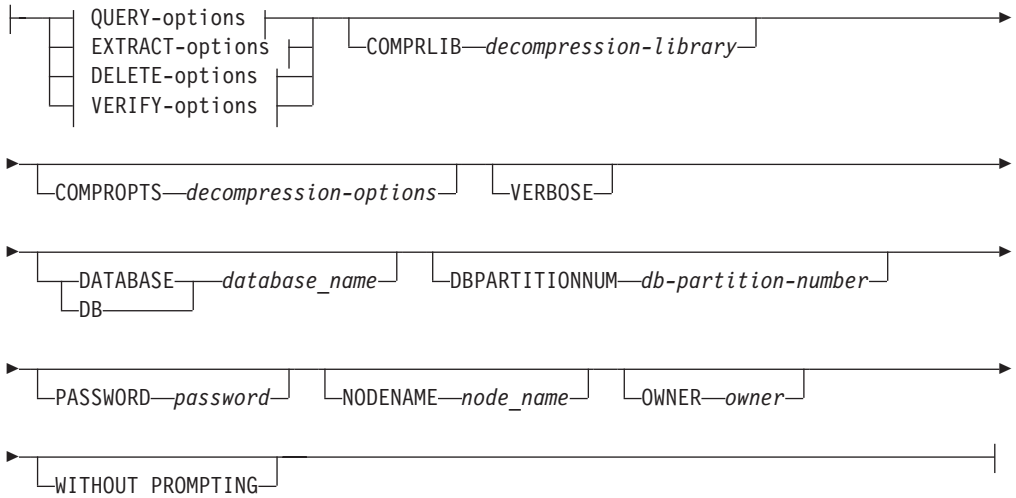
None

Required connection:

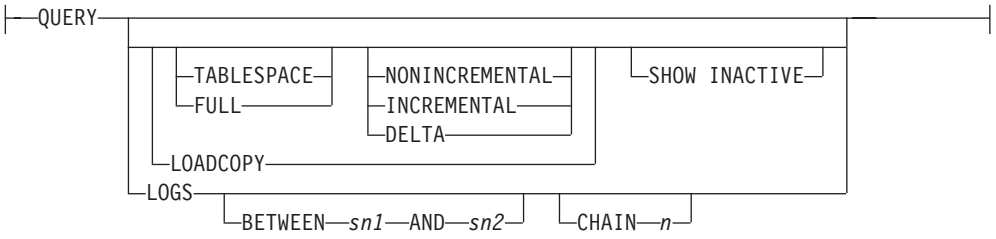
None

Command syntax:

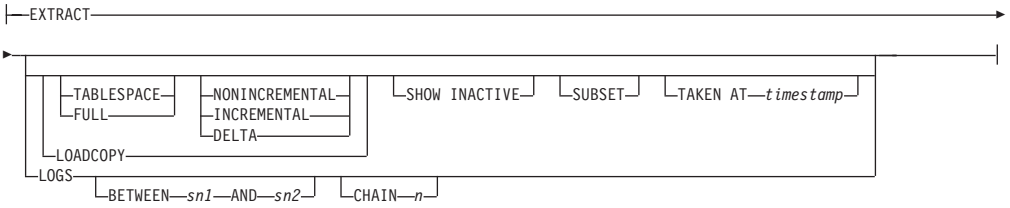
db2-object-options:



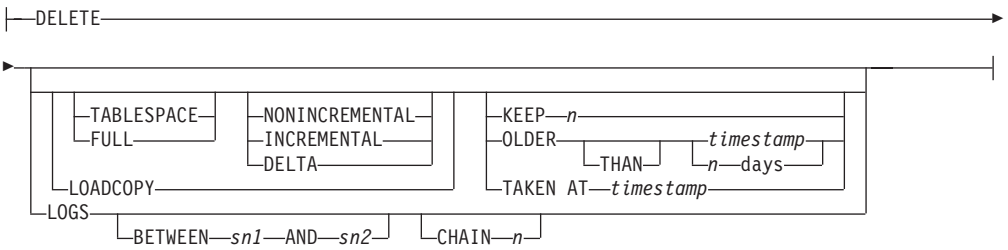
QUERY-options:



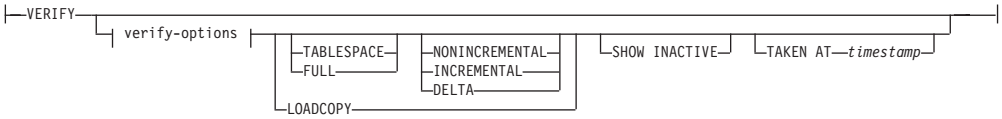
EXTRACT-options:



DELETE-options:

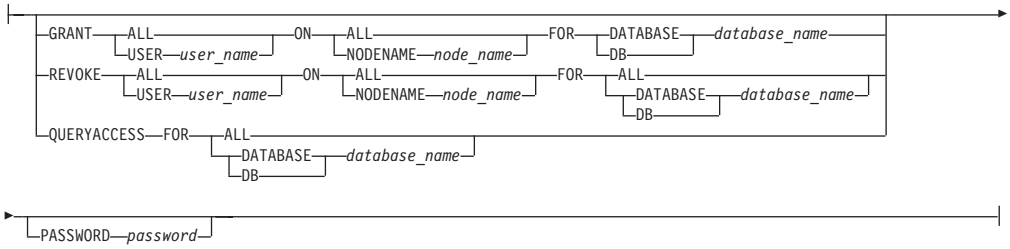


VERIFY-options:



verify-options:

access-control-options:



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. This parameter causes the entire backup image to be transferred over the network.

ALL Displays all available information.

CHECK
Displays results of checkbits and checksums.

DMS Displays information from headers of DMS table space data pages.

HEADER
Displays the media header information.

HEADERONLY
Displays the same information as HEADER but only reads the 4 K media header information from the beginning of the image. It does not validate the image.

LFH Displays the log file header (LFH) data.

OBJECT
Displays detailed information from the object headers.

PAGECOUNT
Displays the number of pages of each object type found in the image.

SGF Displays the automatic storage paths in the image.

SGFONLY
Displays only the automatic storage paths in the image but does not validate the image.

TABLESPACES
Displays the table space details, including container information, for the table spaces in the image.

TABLESPACESONLY
Displays the same information as TABLESPACES but does not validate the image.

TABLESPACE
Includes only table space backup images.

FULL Includes only full database backup images.

NONINCREMENTAL
Includes only non-incremental backup images.

INCREMENTAL
Includes only incremental backup images.

DELTA
Includes 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.

CHAIN *n*
Specifies the chain ID of the logs to be used.

SHOW INACTIVE
Includes backup objects that have been deactivated.

SUBSET
Extracts pages from an image to a file. To extract pages, you will need an input and an output file. The default input file is called extractPage.in. You can override the default input file name by setting the DB2LISTFILE environment variable to a full path. The format of the input file is as follows:

For SMS table spaces:

```
S <tbspID> <objID> <objType> <startPage> <numPages>
```

Notes:

1. <startPage> is an object page number that is object-relative.

For DMS table spaces:

```
D <tbspID> <objType> <startPage> <numPages>
```

Notes:

1. <objType> is only needed if verifying DMS load copy images.
2. <startPage> is an object page number that is pool-relative.

For log files:

```
L <log num> <startPos> <numPages>
```

For other data (for example, initial data):

```
O <objType> <startPos> <numBytes>
```

The default output file is extractPage.out. You can override the default output file name by setting the DB2EXTRACTFILE environment variable to a full path.

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.

COMPRLIB *decompression-library*
Indicates the name of the library to be used to perform the decompression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the operation will fail.

COMPROPTS *decompression-options*
Describes a block of binary data that will be passed to the initialization routine in the decompression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the decompression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of the data block with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for this string is 1024 bytes.

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.

OWNER *owner*
Considers only those objects created by the specified owner.

WITHOUT PROMPTING
The user is not prompted for verification before objects are deleted.

VERBOSE
Displays additional file information.

GRANT ALL / USER *user_name*
Adds access rights to the TSM files on the current TSM node to all users or to the users specified. Granting access to users gives them access for all current and future files related to the database specified.

REVOKE ALL / USER *user_name*
Removes access rights to the TSM files on the current TSM node from all users or to the users specified.

QUERYACCESS
Retrieves the current access list. A list of users and TSM nodes is displayed.

ON ALL / NODENAME *node_name*
Specifies the TSM node for which access rights will be changed.

FOR ALL / DATABASE *database_name*
Specifies the database to be considered.

Examples:

1. The following is sample output from the command `db2 backup database rawsampl use tsm`

```
Backup successful. The timestamp for this backup is : 20031209184503
```

The following is sample output from the command `db2adutl query` issued following the backup operation:

```
Query for database RAWSAMPL

Retrieving FULL DATABASE BACKUP information.
    1 Time: 20031209184403, Oldest log: S0000050.LOG, Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for RAWSAMPL

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for RAWSAMPL

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for RAWSAMPL

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for RAWSAMPL

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for RAWSAMPL

Retrieving LOCAL COPY information.
  No LOCAL COPY images found for RAWSAMPL

Retrieving log archive information.
   Log file: S0000050.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.46.13
   Log file: S0000051.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.46.43
   Log file: S0000052.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.47.12
   Log file: S0000053.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.50.14
   Log file: S0000054.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.50.56
   Log file: S0000055.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.52.39
```

2. The following is sample output from the command `db2adutl delete full taken at 20031209184503 db rawsampl`

```
Query for database RAWSAMPL

Retrieving FULL DATABASE BACKUP information.
  Taken at: 20031209184503  DB Partition Number: 0    Sessions: 1
```

```
  Do you want to delete this file (Y/N)? y

    Are you sure (Y/N)? y


Retrieving INCREMENTAL DATABASE  BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for RAWSAMPL

Retrieving DELTA DATABASE  BACKUP information.
  No DELTA DATABASE BACKUP images found for RAWSAMPL
```

The following is sample output from the command `db2adutl query` issued following the operation that deleted the full backup image. Note the timestamp for the backup image.

```
Query for database RAWSAMPL

Retrieving FULL DATABASE BACKUP information.
    1 Time: 20031209184403, Oldest log: S0000050.LOG, Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for RAWSAMPL

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for RAWSAMPL

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for RAWSAMPL

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for RAWSAMPL

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for RAWSAMPL

Retrieving LOCAL COPY information.
  No LOCAL COPY images found for RAWSAMPL

Retrieving log archive information.
   Log file: S0000050.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.46.13
   Log file: S0000051.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.46.43
   Log file: S0000052.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.47.12
   Log file: S0000053.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.50.14
   Log file: S0000054.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.50.56
   Log file: S0000055.LOG, Chain Num: 0, DB Partition Number: 0,
    Taken at 2003-12-09-18.52.39
```

3. The following is sample output from the command `db2adutl queryaccess for all`

```
Node                User                Database Name      type
------------------------------------------------------------------------
bar2                jchisan             sample             B
<all>               <all>               test               B
------------------------------------------------------------------------
Access Types: B – Backup images  L – Logs  A - both
```

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.

Cumulativeness:

- 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.

Related concepts:

- Appendix F, "Cross-node recovery with the db2adutl command and the logarchopt1 and vendoropt database configuration parameters," on page 397

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 metadata stored in the backup header.

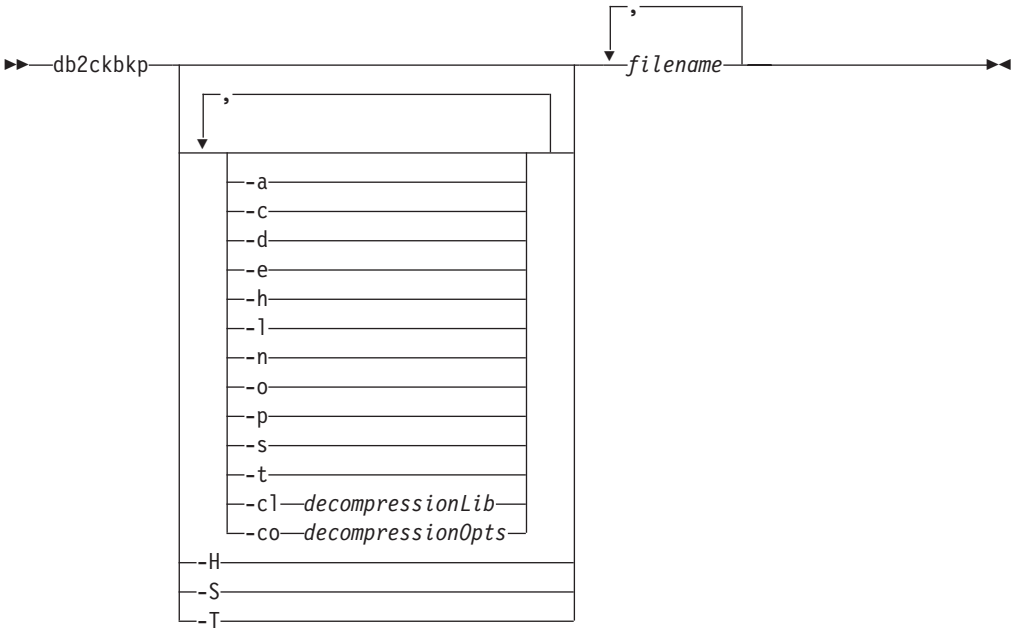
Authorization:

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:



Command parameters:

-a Displays all available information.

-c Displays results of checkbits and checksums.

-cl *decompressionLib*
Indicates the name of the library to be used to perform the decompression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the operation will fail.

-co *decompressionOpts*
Describes a block of binary data that will be passed to the initialization routine in the decompression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the decompression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of *string* with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for string is 1024 bytes.

-d Displays information from the headers of DMS table space data pages.

-e Extracts pages from an image to a file. To extract pages, you will need an input and an output file. The default input file is called extractPage.in. You can override the default input file name by setting the DB2LISTFILE environment variable to a full path. The format of the input file is as follows:

For SMS table spaces:

```
S <tbspID> <objID> <objType> <startPage> <numPages>
```

Notes:

1. <startPage> is an object page number that is object-relative.

For DMS table spaces:

```
D <tbspID> <objType> <startPage> <numPages>
```

Notes:

1. <objType> is only needed if verifying DMS load copy images.
2. <startPage> is an object page number that is pool-relative.

For log files:

```
L <log num> <startPos> <numPages>
```

For other data (for example, initial data):

```
O <objType> <startPos> <numBytes>
```

The default output file is extractPage.out. You can override the default output file name by setting the DB2EXTRACTFILE environment variable to a full path.

-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. This option cannot be used in combination with any other options.

-l Displays log file header (LFH) and mirror log file header (MFH) data.

-n Prompt for tape mount. Assume one tape per device.

-o Displays detailed information from the object headers.

-p Displays the number of pages of each object type. This option will not show the number of pages for all different object types if the backup was done for DMS tablespaces data. It only shows the total of all pages as `SQLUDMSTABLESPACEDATA`. The object types for `SQLUDMSLOBDATA` and `SQLUDMSLONGDATA` will be zero for DMS tablespaces.

-s Displays the automatic storage paths in the image.

-S Displays the same information as `-s` but does not validate the image. This option cannot be used in combination with any other options.

-t Displays table space details, including container information, for the table spaces in the image.

-T Displays the same information as -t but does not validate the image. This option cannot be used in combination with any other options.

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:

Example 1 (on UNIX platforms)

```
db2ckbkp SAMPLE.0.krodger.NODE0000.CATN0000.19990817150714.001
SAMPLE.0.krodger.NODE0000.CATN0000.19990817150714.002
SAMPLE.0.krodger.NODE0000.CATN0000.19990817150714.003

[1] Buffers processed:  ##
[2] Buffers processed:  ##
[3] Buffers processed:  ##
Image Verification Complete - successful.
```

Example 2

```
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
        Include Logs                   -- 0
        Compression                    -- 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                   --
        LogID                          -- 1074717952
        LogPath                       -- /home/krodger/krodger/NODE0000/
                                         SQL00001/SQLOGDIR
        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 - Managing DB2 objects within TSM" on page 303

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.

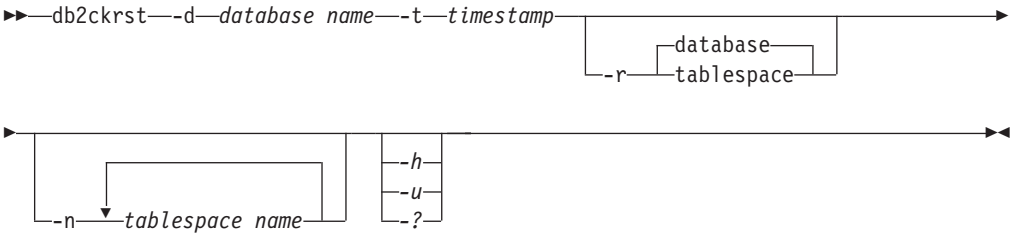
Authorization:

None

Required connection:

None

Command syntax:



Command parameters:

-d *database name*
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. 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. If a database restore type is selected and a list of table space names is specified, the utility will continue as a table space 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 **db2ckrst** utility will not be enhanced for the rebuilding of a database. Due to the constraints of the history file, the utility will not be able to supply the correct list if several table spaces need to be restored from more than one image.

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 can delete entries that are required for automatic incremental restoration of databases. Manual restores will still work correctly. Use of this command can also prevent the dbckrst utility from being able to correctly analyse the complete chain of required backup images. 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.

Related tasks:

- "Restoring from incremental backup images" on page 28

Related reference:

- "PRUNE HISTORY/LOGFILE " on page 329
- "RESTORE DATABASE " on page 100

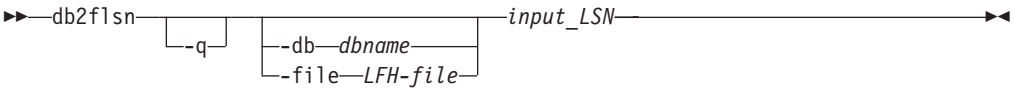
db2flsn - Find log sequence number

Returns the name of the file that contains the log record identified by a specified log sequence number (LSN).

Authorization:

None

Command syntax:



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
- -106 Invalid database
- -500 Logical error

Other valid return codes are:

- 0 Successful execution
- 99 Warning: the result is based on the last known log file size.

-db *dbname*
Specifies the database name which you want to investigate.

-file *LFH-name*
Specifies the full path of the LFH file including the file name.

input_LSN
A 12 or 16 character string that represents the internal (6 or 8 byte) hexadecimal value with leading zeros.

Examples:

```
db2flsn 000000BF0030
   Given LSN is contained in log page 2 in log file S0000002.LOG

db2flsn -q 000000BF0030
   S0000002.LOG

db2flsn 000000BE0030
   Given LSN is contained in log page 2 in log file S0000001.LOG

db2flsn -q 000000BE0030
   S0000001.LOG

db2flsn -db flsntest 0000000000FA0000
   Warning: the result is based on the last known log file size (6
   4K pages starting from log extent 10).  The input_LSN might be before
   the database becomes recoverable.

   Given LSN is contained in log page 2 in log file S0000002.LOG

db2flsn -q -db flsntest 0000000000FA0000
   S0000002.LOG

db2flsn -file C:\DB2\NODE0000\SQL00001\SQLOGCTL.LFH 0000000000FA4368
   Given LSN is contained in log page 6 in log file S0000002.LOG
```

Usage notes:

- If neither `-db` nor `-file` are specified, the tool assumes the LFH file is `SQLOGCTL.LFH` in the current directory.
- 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 the *logarchmeth1* or *logarchmeth2* configuration parameters set to a value other than `OFF`.

Related reference:

- "DB2 log records" in *Administrative API Reference*
- "SQLU_LSN " on page 366

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. This command can only be run against a split mirror database, and it must be run before the split mirror can be used.

Authorization:

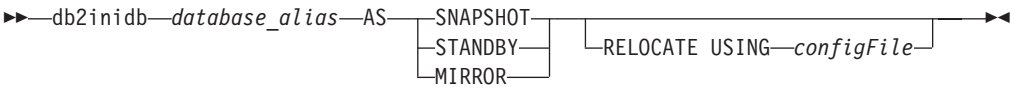
One of the following:

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

Required connection:

None

Command syntax:



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. 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 specified *configFile* prior to initializing the database as a snapshot, standby, or mirror. The format of *configFile* is described in db2relocatedb - Relocate database command.

Usage notes:

Do not issue the **db2 connect to <database>** command before issuing the **db2init <database> as mirror** command. Attempting to connect to a split mirror database before initializing it erases the log files needed during roll forward recovery. The connect sets your database back to the state it was in when you suspended the database. If the database is marked as consistent when it was suspended, the DB2 database system concludes there is no need for crash recovery and empties the logs for future use. If the logs have been emptied, attempting to roll forward results in the SQL4970N error message being returned.

In a partitioned database environment, **db2inidb** must be run on every database partition before the split mirror from any of the database partitions can be used. **db2inidb** can be run on all database partitions simultaneously using the **db2_all** command.

If; however, you are using the RELOCATE USING option, you cannot use the **db2_all** command to run **db2inidb** on all of the partitions simultaneously. A separate configuration file must be supplied for each partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the **db2relocatedb** command must be run with a separate configuration file on

each database partition. If containers belonging to a single database partition are being moved, the **db2relocatedb** command only needs to be run once on that database partition.

If the RELOCATE USING *configFile* parameter is specified and the database is relocated successfully, the specified *configFile* will be copied into the database directory and renamed to `db2path.cfg`. During a subsequent crash recovery or rollforward recovery, this file will be used to rename container paths as log files are being processed.

If a clone database is being initialized, the specified *configFile* will be automatically removed from the database directory after a crash recovery is completed.

If a standby database or mirrored database is being initialized, the specified *configFile* will be automatically removed from the database directory after a rollforward recovery is completed or canceled. New container paths can be added to the `db2path.cfg` file after **db2inidb** has been run. This would be necessary when CREATE or ALTER TABLESPACE operations are done on the original database and different paths must be used on the standby database.

Related tasks:

- "Using a split mirror to clone a database" on page 212
- "Using a split mirror as a backup image" on page 214
- "Using a split mirror as a standby database" on page 213

Related reference:

- "db2relocatedb - Relocate database command" in *Command Reference*
- "rah and db2_all command descriptions" in *Administration Guide: Implementation*

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:

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 user name of the domain account for the DB2 service (specified as *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 database 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 database 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. 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 (for example, 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.

Related reference:

- "db2drvmp - DB2 database drive map command" in *Command Reference*

db2rfpen - Reset rollforward pending state

Puts a database in rollforward pending state. If you are using high availability disaster recovery (HADR), the database is reset to a standard database.

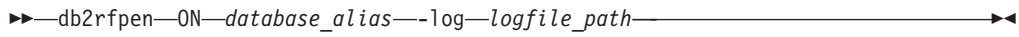
Authorization:

None

Required connection:

None

Command syntax:



Command parameters:

database_alias
Specifies the name of the database to be placed in rollforward pending state. If you are using high availability disaster recovery (HADR), the database is reset to a standard database.

-log logfile_path
Specifies the log file path.

Related concepts:

- "High availability disaster recovery overview" on page 221

CLP commands

ARCHIVE LOG

Closes and truncates the active log file for a recoverable database.

Authorization:

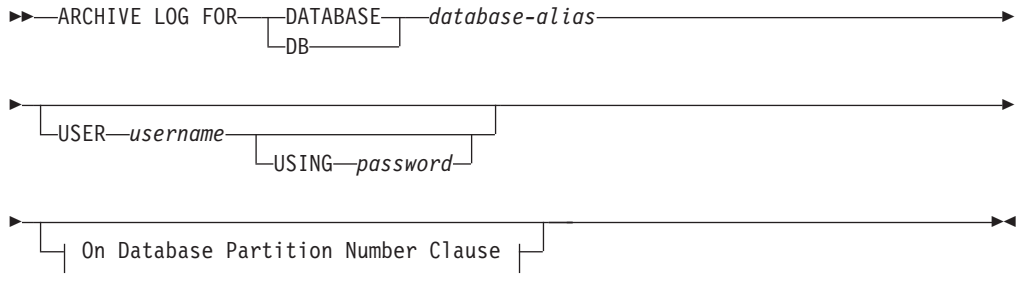
One of the following:

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

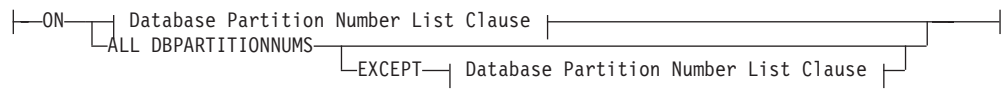
Required connection:

None. This command establishes a database connection for the duration of the command.

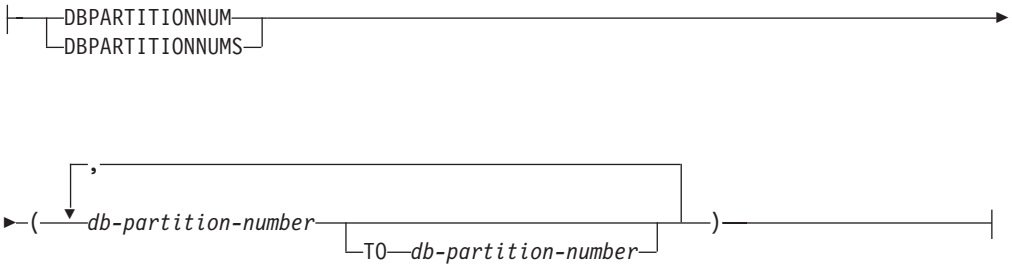
Command syntax:



On Database Partition Number Clause:



Database Partition Number List Clause:



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 can be specified by using a database partition number clause. If the database partition number clause is not specified, the default behavior 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 can 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.

Related reference:

- "db2ArchiveLog - Archive the active log file" on page 335

INITIALIZE TAPE

Initializes tapes for backup and restore operations to streaming tape devices. This command is only supported on Windows operating systems.

Authorization:

One of the following:

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

Required connection:

None.

Command syntax:

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.

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.

When backing up to tape, 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.

Related reference:

- "RESTORE DATABASE " on page 100
- "BACKUP DATABASE " on page 71
- "REWIND TAPE " on page 330
- "SET TAPE POSITION " on page 331
- "INITIALIZE TAPE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

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.

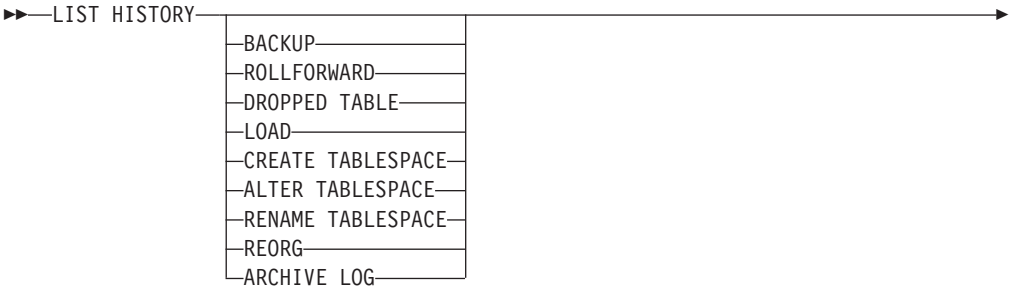
Authorization:

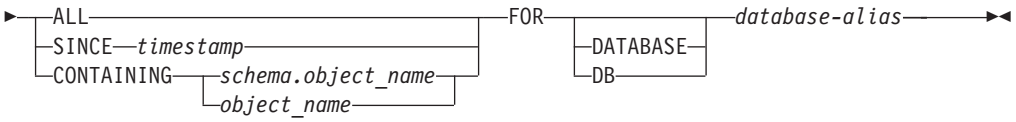
None

Required connection:

Instance. You must attach to any remote database in order to run this command against it. For a local database, an explicit attachment is not required.

Command syntax:





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. A dropped table record is created only when the table is dropped and the table space containing it has the DROPPED TABLE RECOVERY option enabled. Returns the CREATE TABLE syntax for partitioned tables and indicates which table spaces contained data for the table that was dropped.

LOAD
Lists load operations.

CREATE TABLESPACE
Lists table space create and drop operations.

RENAME TABLESPACE
Lists table space renaming operations.

REORG
Lists reorganization operations. Includes information for each reorganized data partition of a partitioned table.

ALTER TABLESPACE
Lists alter table space operations.

ARCHIVE LOG
Lists archive log operations and the archived logs.

ALL Lists all entries of the specified type in the history file.

SINCE timestamp
A complete time stamp (format *yyyymmddhhmmss*), 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 SYSIBMADM.DB_HISTORY administrative view can be used to retrieves data from all database partitions.

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
   X - Archive log

Type

Archive Log types:
   P - Primary log path
   M - Secondary (mirror) log path
   N - Archive log command
   F - Failover archive path
   1 - Primary log archive method
   2 - Secondary log archive method

Backup types:

   F - Offline
   N - Online
   I - Incremental offline
   O - Incremental online
   D - Delta offline
   E - Delta online
   R - Rebuild

Rollforward types:

   E - End of logs
   P - Point in time

Load types:

   I - Insert
   R - Replace

Alter table space types:

   C - Add containers
   R - Rebalance

Quiesce types:

   S - Quiesce share
   U - Quiesce update
   X - Quiesce exclusive
   Z - Quiesce reset
```

Related concepts:

• "Developing a backup and recovery strategy" on page 3

Related reference:

- "DB_HISTORY administrative view – Retrieve history file information" in *Administrative SQL Routines and Views*

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 might be necessary if the file becomes excessively large and the retention period is high.

Authorization:

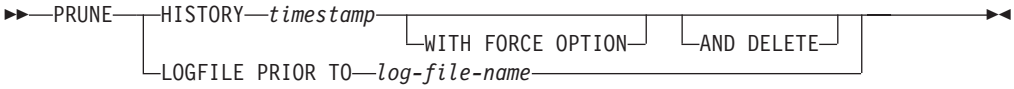
One of the following:

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

Required connection:

Database

Command syntax:



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.

AND DELETE
Specifies that the associated log archives will be physically deleted (based on the location information) when the history file entry is removed. This option is especially useful for ensuring that archive storage space is recovered when log archives are no longer needed. If you are archiving logs via a user exit program, the logs cannot be deleted using this option.

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
```

199412 is interpreted as 19941201000000.

Usage notes:

If the FORCE option is used, you might delete entries that are required for automatic restoration of databases. Manual restores will still work correctly. Use of this command can also prevent the **dbckrst** utility from being able to correctly analyze the complete chain of required backup images. Using the PRUNE HISTORY command without the FORCE option prevents required entries from being deleted.

Pruning backup entries from the history file causes related file backups on DB2 Data Links Manager servers to be deleted.

Related concepts:

- "Developing a backup and recovery strategy" on page 3

Related reference:

- "PRUNE HISTORY/LOGFILE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

REWIND TAPE

Rewinds tapes for backup and restore operations to streaming tape devices. This command is only supported on Windows operating systems.

Authorization:

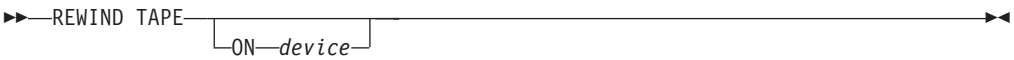
One of the following:

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

Required connection:

None.

Command syntax:



Command parameters:

ON device
Specifies a valid tape device name. The default value is `\\.\TAPE0`.

Related reference:
- "INITIALIZE TAPE " on page 325
- "SET TAPE POSITION " on page 331
- "REWIND TAPE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

SET TAPE POSITION

Sets the positions of tapes for backup and restore operations to streaming tape devices. This command is only supported on Windows operating systems.

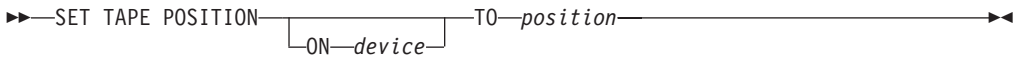
Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None.

Command syntax:



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 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:
- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "SET TAPE POSITION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "INITIALIZE TAPE " on page 325
- "REWIND TAPE " on page 330

UPDATE HISTORY

Updates the location, device type, comment, or status in a history file entry.

Authorization:

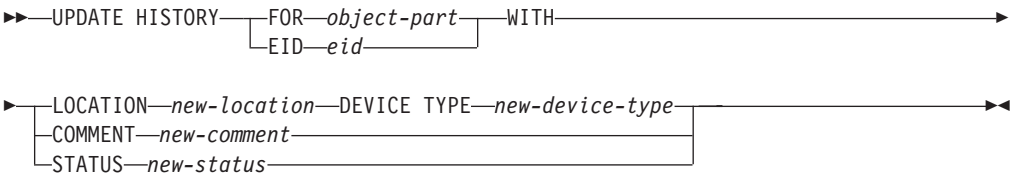
One of the following:

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

Required connection:

Database

Command syntax:



Command parameters:

FOR *object-part*
Specifies the identifier for the history entry to be updated. It is a time stamp with an optional sequence number from `001` to `999`. This parameter cannot be used to update the entry status. To update the entry status, specify an EID instead.

EID *eid*
Specifies the history entry ID.

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.

STATUS *new-status*
Specifies a new status for an entry. Only backup entries can have their status updated. Valid values are:

A Active. Most entries are active.

I Inactive. Backup images that are no longer on the active log chain become inactive.

E Expired. Backup images that are no longer required because there are more than NUM_DB_BACKUPS active images are flagged as expired.

D Deleted. Backup images that are no longer available for recovery should be marked as having been deleted.

Example:

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 primary purpose of the database history file is to record information, but the data contained in the history is used directly by automatic restore operations. During any restore where the AUTOMATIC option is specified, the history of backup images and their locations will be referenced and used by the restore utility to fulfill the automatic restore request. If the automatic restore function is to be used and backup images have been relocated since they were created, it is recommended that the database history record for those images be updated to reflect the current location. If the backup image location in the database history is not updated, automatic restore will not be able to locate the backup images, but manual restore commands can still be used successfully.

Related concepts:

- "Developing a backup and recovery strategy" on page 3

Related reference:

- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "UPDATE HISTORY command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

UPDATE HISTORY

Appendix D. Additional APIs and associated data structures

This appendix describes recovery-related APIs and their data structures that are not discussed in detail in this manual.

db2ArchiveLog - Archive the active log file

Closes and truncates the active log file for a recoverable database. If user exit is enabled, it also issues an archive request.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint
- dbadm

Required connection:

This API automatically establishes a connection to the specified database. If a connection to the specified database already exists, the API will return an error.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2ArchiveLog (
       db2Uint32 versionNumber,
       void * pDB2ArchiveLogStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ArchiveLogStruct
{
   char *piDatabaseAlias;
   char *piUserName;
   char *piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iOptions;
} db2ArchiveLogStruct;

SQL_API_RC SQL_API_FN
  db2gArchiveLog (
       db2Uint32 versionNumber,
       void * pDB2ArchiveLogStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gArchiveLogStruct
{
   db2Uint32 iAliasLen;
   db2Uint32 iUserNameLen;
   db2Uint32 iPasswordLen;
```

```
   char *piDatabaseAlias;
   char *piUserName;
   char *piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iOptions;
} db2gArchiveLogStruct;
```

db2ArchiveLog API parameters:

versionNumber
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.

db2ArchiveLogStruct data structure parameters:

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
Applicable to partitioned database environment only. Input. Flag indicating whether the operation should apply to all nodes listed in the db2nodes.cfg file. Valid values are:

DB2ARCHIVELOG_NODE_LIST
Apply to nodes in a node list that is passed in piNodeList.

DB2ARCHIVELOG_ALL_NODES
Apply to all nodes. piNodeList should be NULL. This is the default value.

DB2ARCHIVELOG_ALL_EXCEPT
Apply to all nodes except those in the node list passed in piNodeList.

iNumNodes
Partitioned database environment only. Input. Specifies the number of nodes in the piNodeList array.

piNodeList
Partitioned database environment only. Input. A pointer to an array of node numbers against which to apply the archive log operation.

iOptions
Input. Reserved for future use.

db2gArchiveLogStruct data structure specific parameters:

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.

Related reference:

- "ARCHIVE LOG " on page 323
- "SQLCA data structure" in *Administrative API Reference*

db2HistoryCloseScan - End the 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 thedb2HistoryOpenScan API.

Authorization:

None

Required connection:

Instance. It is not necessary to call the sqleatin API before calling this API.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HistoryCloseScan (
       db2Uint32 versionNumber,
       void * piHandle,
       struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  db2gHistoryCloseScan (
       db2Uint32 versionNumber,
       void * piHandle,
       struct sqlca * pSqlca);
```

db2HistoryCloseScan API parameters:

versionNumber
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 the db2HistoryOpenScan API.

pSqlca
Output. A pointer to the sqlca structure.

Usage notes:

db2HistoryCloseScan - End the history file scan

For a detailed description of the use of the history file APIs, refer to the db2HistoryOpenScan API.

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.

Related reference:

- "LIST HISTORY " on page 326
- "db2HistoryGetEntry - Get the next entry in the history file" on page 338
- "db2HistoryOpenScan - Start a history file scan" on page 341
- "db2HistoryUpdate - Update a history file entry" on page 345
- "db2Prune - Delete the history file entries or log files from the active log path" on page 348
- "SQLCA data structure" in *Administrative API Reference*

Related samples:

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

db2HistoryGetEntry - Get the next entry in the history file

Gets the next entry from the history file. This API must be preceded by a successful call to the db2HistoryOpenScan API.

Authorization:

None

Required connection:

Instance. It is not necessary to call sqleatin before calling this API.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HistoryGetEntry (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryGetEntryStruct
{
   struct db2HistoryData *pioHistData;
   db2Uint16 iHandle;
   db2Uint16 iCallerAction;
} db2HistoryGetEntryStruct;

SQL_API_RC SQL_API_FN
```

```
db2gHistoryGetEntry (
     db2Uint32 versionNumber,
     void * pParmStruct,
     struct sqlca * pSqlca);
```

db2HistoryGetEntry API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, pParmStruct.

pParmStruct
Input. A pointer to the db2HistoryGetEntryStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2HistoryGetEntryStruct data structure parameters:

pioHistData
Input. A pointer to the db2HistData structure.

iHandle
Input. Contains the handle for scan access that was returned by the db2HistoryOpenScan API.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

Usage notes:

The records that are returned will have been selected using the values specified in the call to the db2HistoryOpenScan API.

For a detailed description of the use of the history file APIs, refer to the db2HistoryOpenScan API.

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).

Related reference:

- "LIST HISTORY " on page 326
- "db2HistoryData " on page 360
- "db2HistoryCloseScan - End the history file scan" on page 337
- "db2HistoryOpenScan - Start a history file scan" on page 341
- "db2HistoryUpdate - Update a history file entry" on page 345
- "db2Prune - Delete the history file entries or log files from the active log path" on page 348
- "SQLCA data structure" in *Administrative API Reference*
- "DB_HISTORY administrative view – Retrieve history file information" in *Administrative SQL Routines and Views*

Related samples:

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

db2HistoryOpenScan - Start a history file scan

This API starts a history file scan.

Authorization:

None

Required connection:

Instance. If the database is cataloged as remote, call the sqleatin API before calling this API.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HistoryOpenScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryOpenStruct
{
   char *piDatabaseAlias;
   char *piTimestamp;
   char *piObjectName;
   db2Uint32 oNumRows;
   db2Uint32 oMaxTbspaces;
   db2Uint16 iCallerAction;
   db2Uint16 oHandle;
} db2HistoryOpenStruct;

SQL_API_RC SQL_API_FN
  db2gHistoryOpenScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHistoryOpenStruct
{
   char *piDatabaseAlias;
   char *piTimestamp;
   char *piObjectName;
   db2Uint32 iAliasLen;
   db2Uint32 iTimestampLen;
   db2Uint32 iObjectNameLen;
   db2Uint32 oNumRows;
   db2Uint32 oMaxTbspaces;
   db2Uint16 iCallerAction;
   db2Uint16 oHandle;
} db2gHistoryOpenStruct;
```

db2HistoryOpenScan API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, pParmStruct.

pParmStruct
Input or Output. A pointer to the db2HistoryOpenStruct data structure.

pSqlca
Output. A pointer to the sqlca structure.

db2HistoryOpenStruct data structure parameters:

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 call, this parameter contains the number of matching history file entries.

oMaxTbspaces
Output. The maximum number of table space names stored with any history entry.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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 the db2HistoryGetEntry, and db2HistoryCloseScan APIs.

db2gHistoryOpenStruct data structure specific parameters:

iAliasLen
Input. Specifies the length in bytes of the database alias string.

iTimestampLen
Input. Specifies the length in bytes of the timestamp string.

iObjectNameLen
Input. Specifies the length in bytes of the object name string.

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 the db2HistoryOpenScan API, which returns parameter value oNumRows.
2. Allocate a db2HistData structure with space for n oTablespace fields, where n is an arbitrary number.
3. Set the iNumTablespaces field of the db2HistoryData structure to n.
4. In a loop, perform the following:
   - Call the db2HistoryGetEntry API to fetch from the history file.
   - If db2HistoryGetEntry API returns an SQLCODE value of SQL_RC_OK, use the oNumTablespaces field of the db2HistoryData structure to determine the number of table space entries returned.
   - If db2HistoryGetEntry API returns an SQLCODE value 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 db2HistoryData structure with enough space for oDB2UsedTablespace table space entries, and set iDB2NumTablespace to oDB2UsedTablespace.
   - If db2HistoryGetEntry API returns an SQLCODE value 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 the db2HistoryCloseScan API to free the resources allocated by the call to db2HistoryOpenScan.

The macro SQLUHINFOSIZE(n) (defined in sqlutil header file) is provided to help determine how much memory is required for a db2HistoryData structure with space for n oTablespace entries.

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 that is to have its history file 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.

Related reference:

- "LIST HISTORY " on page 326
- "db2HistoryCloseScan - End the history file scan" on page 337
- "db2HistoryGetEntry - Get the next entry in the history file" on page 338
- "db2HistoryUpdate - Update a history file entry" on page 345
- "db2Prune - Delete the history file entries or log files from the active log path" on page 348
- "SQLCA data structure" in *Administrative API Reference*

Related samples:

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

db2HistoryUpdate - Update a history file entry

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
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2HistoryUpdate (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryUpdateStruct
{
   char *piNewLocation;
   char *piNewDeviceType;
   char *piNewComment;
   char *piNewStatus;
   db2HistoryEID iEID;
} db2HistoryUpdateStruct;

typedef SQL_STRUCTURE db2HistoryEID
{
   SQL_PDB_NODE_TYPE ioNode;
   db2Uint32 ioHID;
} db2HistoryEID;

SQL_API_RC SQL_API_FN
  db2gHistoryUpdate (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHistoryUpdateStruct
{
   char *piNewLocation;
   char *piNewDeviceType;
   char *piNewComment;
   char *piNewStatus;
   db2Uint32 iNewLocationLen;
   db2Uint32 iNewDeviceLen;
   db2Uint32 iNewCommentLen;
   db2Uint32 iNewStatusLen;
   db2HistoryEID iEID;
} db2gHistoryUpdateStruct;
```

db2HistoryUpdate API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, pParmStruct.

pParmStruct
Input. A pointer to the db2HistoryUpdateStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2HistoryUpdateStruct data structure parameters:

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.

piNewStatus
Input. A pointer to a string specifying a new status type for the entry. Setting this parameter to NULL, or pointing to zero, leaves the status unchanged.

iEID Input. A unique identifier that can be used to update a specific entry in the history file.

db2HistoryEID data structure parameters:

ioNode
This parameter can be used as either an input or output parameter.

Indicates the node number.

ioHID This parameter can be used as either an input or output parameter.

Indicates the local history file entry ID.

db2gHistoryUpdateStruct data structure specific parameters:

iNewLocationLen
Input. Specifies the length in bytes of the piNewLocation parameter.

iNewDeviceLen
Input. Specifies the length in bytes of the piNewDeviceType parameter.

iNewCommentLen
Input. Specifies the length in bytes of the piNewComment parameter.

iNewStatusLen
Input. Specifies the length in bytes of the piNewStatus paramter.

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 primary purpose of the database history file is to record information, but the data contained in the history is used directly by automatic restore operations. During any restore where the AUTOMATIC option is specified, the history of backup images and their locations will be referenced and used by the restore utility to fulfill the automatic restore request. If the automatic restore function is to be used and backup images have been relocated since they were created, it is recommended that the database history record for those images be updated to reflect the current location. If the backup image location in the database history is not updated, automatic restore will not be able to locate the backup images, but manual restore commands can still be used successfully.

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).

Related reference:

- "db2Backup - Back up a database or table space" on page 76
- "db2HistoryCloseScan - End the history file scan" on page 337
- "db2HistoryGetEntry - Get the next entry in the history file" on page 338
- "db2HistoryOpenScan - Start a history file scan" on page 341
- "db2Prune - Delete the history file entries or log files from the active log path" on page 348
- "db2Rollforward - Roll forward a database" on page 177
- "SQLCA data structure" in *Administrative API Reference*
- "UPDATE HISTORY " on page 332

Related samples:

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

db2Prune - Delete the history file entries or log files from the active log path

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
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Prune (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2PruneStruct
{
  char *piString;
  db2HistoryEID iEID;
  db2Uint32 iAction;
  db2Uint32 iOptions;
} db2PruneStruct;

SQL_API_RC SQL_API_FN
  db2gPrune (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gPruneStruct
{
  db2Uint32 iStringLen;
  char *piString;
  db2HistoryEID iEID;
  db2Uint32 iAction;
  db2Uint32 iOptions;
} db2gPruneStruct;
```

db2Prune API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, pParmStruct.

pParmStruct
Input. A pointer to the db2PruneStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2PruneStruct data structure parameters:

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; a NULL parameter value is invalid.

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.

iAction
Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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 header file, located in the include directory) are:

DB2PRUNE_OPTION_FORCE
Force the removal of the last backup.

DB2PRUNE_OPTION_DELETE
Delete log files that are pruned from the history file.

DB2PRUNE_OPTION_LSNSTRING
Specify that the value of piString is an LSN, used when a caller action of DB2PRUNE_ACTION_LOG is specified.

db2gPruneStruct data structure specific parameters:

iStringLen
Input. Specifies the length in bytes of piString.

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.

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.

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.

Related reference:

- "PRUNE HISTORY/LOGFILE " on page 329
- "db2HistoryCloseScan - End the history file scan" on page 337
- "db2HistoryGetEntry - Get the next entry in the history file" on page 338
- "db2HistoryOpenScan - Start a history file scan" on page 341
- "db2HistoryUpdate - Update a history file entry" on page 345
- "SQLCA data structure" in *Administrative API Reference*
- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*

Related samples:

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

db2ReadLogNoConn - Read the database logs without a database connection

Extracts log records from the DB2 database logs and queries the Log Manager for current log state information. Prior to using this API, call the db2ReadLogNoConnInit API to allocate the memory that is passed as an input parameter to this API. After calling this API, call the db2ReadLogNoConnTerm API to deallocate the memory.

Authorization:

None

Required connection:

None

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
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;
```

db2ReadLogNoConn API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter, pDB2ReadLogNoConnStruct.

pDB2ReadLogNoConnStruct
Input. A pointer to the db2ReadLogNoConnStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ReadLogNoConnStruct data structure parameters:

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.

db2ReadLogNoConnInfoStruct data structure parameters:

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.

currentTimeValue
Reserved for future use.

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 log records returned by db2ReadLogNoConn can be found in the DB2 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.

This API reads data from the DB2 logs. Label-based access control (LBAC) is not enforced on such logs. Thus, an application that calls this API can potentially gain access to table data if the caller has sufficient authority to call the API and is able to understand the log records format.

Related reference:

- "db2ReadLogNoConnInit - Initialize reading the database logs without a database connection" on page 353
- "db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection" on page 355
- "DB2 log records" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "db2ReadLog - Extracts log records" on page 356

db2ReadLogNoConnInit - Initialize reading the database logs without a database connection

Allocates the memory to be used by db2ReadLogNoConn in order to extract log records from the DB2 database logs and query the Log Manager for current log state information.

Authorization:

None

Required connection:

None

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
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;
```

db2ReadLogNoConnInit API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2ReadLogNoConnInitStruct.

pDB2ReadLogNoConnInitStruct
Input. A pointer to the db2ReadLogNoConnInitStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ReadLogNoConnInitStruct data structure parameters:

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 the database logs without a database connection" on page 350
- "db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection" on page 355
- "DB2 log records" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection

Deallocates the memory used by the db2ReadLogNoConn API, originally initialized by the db2ReadLogNoConnInit API.

Authorization:

None

Required connection:

None

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2ReadLogNoConnTerm (
       db2Uint32 versionNumber,
       void * pDB2ReadLogNoConnTermStruct,
       struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
   char                                **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
```

db2ReadLogNoConnTerm API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2ReadLogNoConnTermStruct.

pDB2ReadLogNoConnTermStruct
Input. A pointer to the db2ReadLogNoConnTermStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ReadLogNoConnTermStruct data structure parameters:

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 the database logs without a database connection" on page 350
- "db2ReadLogNoConnInit - Initialize reading the database logs without a database connection" on page 353
- "DB2 log records" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

db2ReadLog - Extracts log records

Extracts log records from the DB2 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 the database configuration parameters logarchmeth1 and/or logarchmeth2 are not set to OFF.

Authorization:

One of the following:

- sysadm
- dbadm

Required connection:

Database

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
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;
} db2ReadLogStruct;

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;
```

db2ReadLog 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 structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ReadLogStruct data structure parameters:

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 the startLsn parameter, 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 behavior of the asynchronous log read API. The log records that are returned when this value is used are documented in the ″DB2 log records″ topic. All other log records are for IBM internal use only and are therefore not documented.

poReadLogInfo
Output. A structure detailing information regarding the call and the database log.

db2ReadLogInfoStruct data structure parameters:

initialLSN
The first LSN used, or that will be used, by the database since it was activated.

firstReadLSN
The first LSN present in poLogBuffer parameter.

nextStartLSN
The start of the next log record the caller should read. Because some log records can be filtered and not returned in poLogBuffer parameter, using this LSN as the start of the next read instead of the end of the last log record in poLogBuffer parameter will prevent rescanning log records which have already been filtered.

logRecsWritten
The number of log records written to poLogBuffer parameter.

logBytesWritten
The total number of bytes of data written to poLogBuffer parameter.

firstReusedLSN
The first LSN to be reused due to a database restore or rollforward operation.

timeOfLSNReuse
The time at which the LSN represented by firstReusedLSN was reused. The time is the number of seconds since January 1, 1970.

currentTimeValue
The current time according to the database.

db2TimeOfLog data structure parameters:

seconds
The number of seconds since Januray 1, 1970.

accuracy
A high accuracy counter which allows callers to distinguish the order of events when comparing timestamps that occurred within the same second.

Usage notes:

If the requested action is to read the log, you must 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 defined with the DATA CAPTURE CHANGES clause, and a db2ReadLogInfoStruct structure with the current active log information. If the requested action is a query of the database log (indicated by specifying the value DB2READLOG_QUERY), the API returns a 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 that are 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 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.

This API reads data from the DB2 logs. Label-based access control (LBAC) is not enforced on such logs. Thus, an application that calls this API can gain access to table data if the caller has sufficient authority to call the API and is able to understand the log records format.

The db2ReadLog API works on the current database connection. If multiple database connections are created with the same process, then use the concurrent access APIs to manage the multiple contexts.

Calling the db2ReadLog API from an application can result in an error when the application disconnects from the database if a commit or rollback is not performed before the disconnect:

- A CLI0116E error might be generated if the db2ReadLog API is called from a CLI application.
- A SQL0428N error might be generated if the db2ReadLog API is called from an embedded SQL application written in C.

Workaround 1: For non-embedded SQL applications, set autocommit mode on before calling the db2ReadLog API.

Workaround 2: Issue a COMMIT or ROLLBACK statement after calling the db2ReadLog API and before disconnecting from the database.

Related reference:

- "db2ReadLogNoConnInit - Initialize reading the database logs without a database connection" on page 353
- "db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection" on page 355
- "DB2 log records" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "db2ReadLogNoConn - Read the database logs without a database connection" on page 350

Related samples:

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

db2HistoryData

This structure is used to return information after a call to the db2HistoryGetEntry API.

Table 4. Fields in the db2HistoryData 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.

Table 4. Fields in the db2HistoryData Structure (continued)

Field Name	Data Type	Description
oObjectPart	db2Char	The first 14 characters are a time stamp with format yyyymmddhhmmss, 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 yyyymmddhhmmss, 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).
oID	db2Char	Unique backup or table identifier.

Table 4. Fields in the db2HistoryData Structure (continued)

Field Name	Data Type	Description
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 parameter 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 parameter: - 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.
iNumTablespaces	db2Uint32	Number of entries in the poTablespace list that are available for use by the db2HistoryGetEntry API.

Table 4. Fields in the db2HistoryData Structure (continued)

Field Name	Data Type	Description
oNumTablespaces	db2Uint32	Number of entries in the poTablespace list that were used by the db2HistoryGetEntry API. 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 following table.
oObject	char	Granularity of the operation: D for full database, P for table space, and T for table.
oOptype	char	See the table titled ″Valid oOptype Values in the db2HistData Structure″.
oStatus	char	Entry status: A for action, D for deleted ,E for expired, I for inactive, N for not yet committed, 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 5. Valid oOperation Values in the db2HistoryData 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

Table 5. Valid oOperation Values in the db2HistoryData Structure (continued)

Value	Description	C Definition	COBOL/FORTRAN Definition
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
X	log archive	DB2HISTORY_OP_ARCHIVE_LOG	DB2HIST_OP_ARCHIVE_LOG

Table 6. Valid oOptype Values in the db2HistData Structure

oOperation	oOptype	Description	C/COBOL/FORTRAN Definition
B	F N I O D E	offline, online, incremental offline, incremental online, delta offline, delta online	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE, DB2HISTORY_OPTYPE_INCR_OFFLINE, DB2HISTORY_OPTYPE_INCR_ONLINE, DB2HISTORY_OPTYPE_DELTA_OFFLINE, DB2HISTORY_OPTYPE_DELTA_ONLINE
F	E P	end of logs, point in time	DB2HISTORY_OPTYPE_EOL, DB2HISTORY_OPTYPE_PIT
G	F N	offline, online	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE
L	I R	insert, replace	DB2HISTORY_OPTYPE_INSERT, DB2HISTORY_OPTYPE_REPLACE
Q	S U X Z	quiesce share, quiesce update, quiesce exclusive, quiesce reset	DB2HISTORY_OPTYPE_SHARE, DB2HISTORY_OPTYPE_UPDATE, DB2HISTORY_OPTYPE_EXCL, DB2HISTORY_OPTYPE_RESET
R	F N I O R	offline, online, incremental offline, incremental online, rebuild	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE, DB2HISTORY_OPTYPE_INCR_OFFLINE, DB2HISTORY_OPTYPE_INCR_ONLINE, DB2HISTORY_OPTYPE_REBUILD
T	C R	add containers, rebalance	DB2HISTORY_OPTYPE_ADD_CONT, DB2HISTORY_OPTYPE_REB
X	N P M F 1 2	archive log command, primary log path, mirror log path, archive fail path, log archive method 1, log archive method 2	DB2HISTORY_OPTYPE_ARCHIVE_CMD, DB2HISTORY_OPTYPE_PRIMARY, DB2HISTORY_OPTYPE_MIRROR, DB2HISTORY_OPTYPE_ARCHFAIL, DB2HISTORY_OPTYPE_ARCH1, DB2HISTORY_OPTYPE_ARCH2

Table 7. Fields in the db2HistoryEID Structure

Field Name	Data Type	Description
ioNode ioHID	SQL_PDB_NODE_TYPE db2Uint32	Node number. Local history file entry ID.

API and data structure syntax:

```
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;
   struct db2Char *poTablespace;
   db2Uint32 iNumTablespaces;
   db2Uint32 oNumTablespaces;
   char oOperation;
   char oObject;
   char oOptype;
   char oStatus;
   char oDeviceType;
} db2HistoryData;

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

typedef SQL_STRUCTURE db2HistoryEID
{
   SQL_PDB_NODE_TYPE ioNode;
   db2Uint32 ioHID;
} db2HistoryEID;
```

db2Char data structure parameters:

pioData
A pointer to a character data buffer. If NULL, no data will be returned.

iLength
Input. The size of the pioData buffer.

oLength
Output. The number of valid characters of data in the pioData buffer.

db2HistoryEID data structure parameters:

ioNode
This parameter can be used as either an input or output parameter. Indicates the node number.

ioHID This parameter can be used as either an input or output parameter. Indicates the local history file entry ID.

Related reference:

- "db2HistoryGetEntry - Get the next entry in the history file" on page 338
- "SQLCA data structure" in *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. An LSN represents the byte offset of the log record from the beginning of the database log.

Table 8. 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.

API and data structure syntax:

```
typedef union SQLU_LSN
{
   unsigned char   lsnChar[6];
   unsigned short  lsnWord[3];
} SQLU_LSN;
```

Related reference:

- "db2ReadLog - Extracts log records" on page 356

Appendix E. Recovery sample programs

Sample programs with embedded SQL

The following sample programs shows how to use DB2 backup and restore APIs to:

- Read and update a database recovery file entry
- Read database log files with a database connection
- Read database log files with no database connection
- Restore a database from a backup image
- Perform a rollforward operation after a database restore operation

Note: These sample files can be found in sqllib/samples/c and sqllib/samples/cpp directory.

dbredirect sample program:

The dbredirect sample files show how to perform a redirected restore of database.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 2003
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
****************************************************************************
**
** SOURCE FILE NAME: dbredirect.sqc
**
** SAMPLE: How to perform Redirected Restore of a database
**
**         This program ends in ".sqc" even though it does not contain
**         embedded SQL statements. It links in  the embedded SQL utility
**         file for database connection and disconnection, so it needs the
**         embedded SQL extension for the precompiler.
**
**         Note:
**           You must be disconnected from the sample database to run
**           this program. To ensure you are, enter 'db2 connect reset'
**           on the command line prior to running dbredirect. If the target
**           database for the redirected restore already exists, SQLCODE 2529
**           will be displayed.
**
** DB2 API USED:
**         db2CfgSet -- Set Configuration
**         db2Restore -- Restore Database
**
** OUTPUT FILE: dbredirect.out (available in the online documentation)
****************************************************************************
**
** For detailed information about database backup and database 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 on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
******************************************************************************/
#include "utilrecov.c"

/* local function prototypes */
int DbBackupAndRedirectedRestore(char *, char *, char *, char *, char *);

/* support function called by DbBackupAndRedirectedRestore() */
int InaccessableContainersRedefine(char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1] = { 0 };
  char serverWorkingPath[SQL_PATH_SZ + 1] = { 0 };
  char redirectedRestoredDbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char dbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char user[USERID_SZ + 1] = { 0 };
  char pswd[PSWD_SZ + 1] = { 0 };

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  CHECKRC(rc, "CmdLineArgsCheck3");

  printf("\nTHIS SAMPLE SHOWS HOW TO PERFORM A REDIRECTED RESTORE\n");
  printf("FROM A DATABASE BACKUP.\n");

  strcpy(redirectedRestoredDbAlias, "RRDB");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  CHECKRC(rc, "Instance Attach");

  /* get the server working path */
  rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
  CHECKRC(rc, "ServerWorkingPathGet");

  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 function */
  rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);
  CHECKRC(rc, "DbRecoveryHistoryFilePrune");

  rc = DbBackupAndRedirectedRestore(dbAlias,
                                    redirectedRestoredDbAlias,
                                    user, pswd, serverWorkingPath);
  CHECKRC(rc, "DbBackupAndRedirectedRestore");

  /* Detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  CHECKRC(rc, "InstanceDetach");

  return 0;
```

```
} /* end main */

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 = 0;

  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1] = { 0 };

  db2BackupStruct backupStruct;
  db2TablespaceStruct tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32 backupImageSize = 0;
  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    */
  /******************************/
```

```
/* Calling up the routine for database backup */
rc = DbBackup(dbAlias, user, pswd, serverWorkingPath, &backupStruct);
CHECKRC(rc, "DbBackup");

/*****************************/
/*    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 = 2;
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);

/* If restoring to a different database and restoreDbAlias already exists,
   SQLCODE 2529 is expected. */
if (strcmp(dbAlias, restoredDbAlias))
{
  printf("\n  SQLCODE 2529 is expected if target database '%s' already exists\n",
        restoredDbAlias);
}

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);
      CHECKRC(rc, "InaccessableContainersRedefine");
    }

    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);
  CHECKRC(rc, "DbDrop");

  return 0;
} /* DbBackupAndRedirectedRestore */

int InaccessableContainersRedefine(char serverWorkingPath[])
{
  struct sqlca sqlca;
  sqluint32 numTablespaces = 0;
  struct SQLB_TBSPQRY_DATA **ppTablespaces = NULL;
  sqluint32 numContainers = 0;
  struct SQLB_TBSCONTQRY_DATA *pContainers = NULL;
  int tspNb = 0;
  int contNb = 0;
  char pathSep[2] = { 0 };

  /* 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 */
```

dbhistfile sample program:

The `dbhistfile` sample files show how to read and update a database recovery file entry.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 2003
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
****************************************************************************
**
** SOURCE FILE NAME: dbhistfile.sqc
**
** SAMPLE: How to read and update a database recovery history file entry.
**
**         This program ends in ".sqc" even though it does not contain
**         embedded SQL statements. It links in  the embedded SQL utility
**         file for database connection and disconnection, so it needs the
**         embedded SQL extension for the precompiler.
**
```

```
** DB2 APIs 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
**
** OUTPUT FILE: dbhistfile.out (available in the online documentation)
******************************************************************************
**
** For more information on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
******************************************************************************/
#include "utilrecov.c"

/* local function prototypes */
int DbRecoveryHistoryFileRead(char *);
int DbFirstRecoveryHistoryFileEntryUpdate(char *, char *, char *);
int HistoryEntryDataFieldsAlloc(struct db2HistoryData *);
int HistoryEntryDisplay(struct db2HistoryData );
int HistoryEntryDataFieldsFree(struct db2HistoryData *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1] = { 0 };
  char serverWorkingPath[SQL_PATH_SZ + 1] = { 0 };
  sqluint16 savedLogRetainValue = 0;
  char dbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char user[USERID_SZ + 1] = { 0 };
  char pswd[PSWD_SZ + 1] = { 0 };

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  CHECKRC(rc, "CmdLineArgsCheck3");

  printf("\nTHIS SAMPLE SHOWS HOW TO READ A DATABASE RECOVERY HISTORY FILE \n");
  printf("AND UPDATE A RECOVERY HISTORY FILE ENTRY. \n");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  CHECKRC(rc, "Instance Attach");

  /* get the server working path */
  rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
  CHECKRC(rc, "ServerWorkingPathGet");

  rc = DbRecoveryHistoryFileRead(dbAlias);
  CHECKRC(rc, "DbRecoveryHistoryFileRead");

  rc = DbFirstRecoveryHistoryFileEntryUpdate(dbAlias, user, pswd);
  CHECKRC(rc, "DbFirstRecoveryHistoryFileEntryUpdate");

  /* Detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  CHECKRC(rc, "InstanceDetach");

  return 0;
```

```
} /* end main */

int DbRecoveryHistoryFileRead(char dbAlias[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint32 numEntries = 0;
  sqluint16 recoveryHistoryFileHandle = 0;
  sqluint32 entryNb = 0;
  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);
  CHECKRC(rc, "HistoryEntryDataFieldsAlloc");

  /*****************************************/
  /* 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++)
  {
    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);
    CHECKRC(rc, "HistoryEntryDisplay");
  }

  /*****************************************/
  /* 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);
  CHECKRC(rc, "HistoryEntryDataFieldsFree");

  return 0;
} /* DbRecoveryHistoryFileRead */

int DbFirstRecoveryHistoryFileEntryUpdate(char dbAlias[], char user[],
                                          char pswd[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint16 recoveryHistoryFileHandle = 0;
  struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
  struct db2HistoryData histEntryData;
  char newLocation[DB2HISTORY_LOCATION_SZ + 1] = { 0 };
  char newComment[DB2HISTORY_COMMENT_SZ + 1] = { 0 };
  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);
  CHECKRC(rc, "HistoryEntryDataFieldsAlloc");

  /*****************************************/
  /* 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);
CHECKRC(rc, "HistoryEntryDisplay");

/* update the first history file entry */
rc = DbConn(dbAlias, user, pswd);
CHECKRC(rc, "DbConn");

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);
CHECKRC(rc, "DbDisconn");

/*****************************************/
/* 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);
  CHECKRC(rc, "HistoryEntryDisplay");

  /*****************************************/
  /* 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);
  CHECKRC(rc, "HistoryEntryDataFieldsFree");

 return 0;
} /* DbFirstRecoveryHistoryFileEntryUpdate */

/*****************************************************************************/
/* HistoryEntryDataFieldsAlloc         */
/* Allocates memory for all the fields in a database recovery history      */
/* file entry                                                                */
/*****************************************************************************/
int HistoryEntryDataFieldsAlloc(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb = 0;

  strcpy(pHistEntryData->ioHistDataID, "SQLUHINF");

  pHistEntryData->oObjectPart.pioData = malloc(DB2HISTORY_OBJPART_SZ + 1);
  pHistEntryData->oObjectPart.iLength = DB2HISTORY_OBJPART_SZ + 1;

  pHistEntryData->oEndTime.pioData = malloc(DB2HISTORY_TIMESTAMP_SZ + 1);
  pHistEntryData->oEndTime.iLength = DB2HISTORY_TIMESTAMP_SZ + 1;

  pHistEntryData->oFirstLog.pioData = malloc(DB2HISTORY_LOGFILE_SZ + 1);
  pHistEntryData->oFirstLog.iLength = DB2HISTORY_LOGFILE_SZ + 1;

  pHistEntryData->oLastLog.pioData = malloc(DB2HISTORY_LOGFILE_SZ + 1);
  pHistEntryData->oLastLog.iLength = DB2HISTORY_LOGFILE_SZ + 1;

  pHistEntryData->oID.pioData = malloc(DB2HISTORY_ID_SZ + 1);
  pHistEntryData->oID.iLength = DB2HISTORY_ID_SZ + 1;

  pHistEntryData->oTableQualifier.pioData =
    malloc(DB2HISTORY_TABLE_QUAL_SZ + 1);
  pHistEntryData->oTableQualifier.iLength = DB2HISTORY_TABLE_QUAL_SZ + 1;

  pHistEntryData->oTableName.pioData = malloc(DB2HISTORY_TABLE_NAME_SZ + 1);
  pHistEntryData->oTableName.iLength = DB2HISTORY_TABLE_NAME_SZ + 1;

  pHistEntryData->oLocation.pioData = malloc(DB2HISTORY_LOCATION_SZ + 1);
  pHistEntryData->oLocation.iLength = DB2HISTORY_LOCATION_SZ + 1;

  pHistEntryData->oComment.pioData = malloc(DB2HISTORY_COMMENT_SZ + 1);
  pHistEntryData->oComment.iLength = DB2HISTORY_COMMENT_SZ + 1;

  pHistEntryData->oCommandText.pioData = malloc(DB2HISTORY_COMMAND_SZ + 1);
  pHistEntryData->oCommandText.iLength = DB2HISTORY_COMMAND_SZ + 1;

  pHistEntryData->poEventSQLCA =
    (struct sqlca *)malloc(sizeof(struct sqlca));
```

```
  pHistEntryData->poTablespace = (db2Char *) malloc(3 * sizeof(db2Char));
  for (tsNb = 0; tsNb < 3; tsNb++)
  {
    pHistEntryData->poTablespace[tsNb].pioData = malloc(18 + 1);
    pHistEntryData->poTablespace[tsNb].iLength = 18 + 1;
  }

  pHistEntryData->iNumTablespaces = 3;

  return 0;
} /* HistoryEntryDataFieldsAlloc */

/******************************************************************************/
/* HistoryEntryDisplay                */
/* Displays the fields of an entry in the database recovery history file   */
/******************************************************************************/
int HistoryEntryDisplay(struct db2HistoryData histEntryData)
{
  int rc = 0;
  int bufLen = 0;
  char *buf = NULL;
  sqluint32 tsNb = 0;

  bufLen =
    MIN(histEntryData.oObjectPart.oLength,
 histEntryData.oObjectPart.iLength);
  buf = malloc(bufLen + 1);
  memcpy(buf, histEntryData.oObjectPart.pioData, bufLen);
  buf[bufLen] = '\0';
  printf("    object part: %s\n", buf);
  free(buf);

  bufLen =
    MIN(histEntryData.oEndTime.oLength, histEntryData.oEndTime.iLength);
  buf = malloc(bufLen + 1);
  memcpy(buf, histEntryData.oEndTime.pioData, bufLen);
  buf[bufLen] = '\0';
  printf("    end time: %s\n", buf);
  free(buf);

  bufLen =
    MIN(histEntryData.oFirstLog.oLength, histEntryData.oFirstLog.iLength);
  buf = malloc(bufLen + 1);
  memcpy(buf, histEntryData.oFirstLog.pioData, bufLen);
  buf[bufLen] = '\0';
  printf("    first log: %s\n", buf);
  free(buf);

  bufLen =
    MIN(histEntryData.oLastLog.oLength, histEntryData.oLastLog.iLength);
  buf = malloc(bufLen + 1);
  memcpy(buf, histEntryData.oLastLog.pioData, bufLen);
  buf[bufLen] = '\0';
  printf("    last log: %s\n", buf);
  free(buf);

  bufLen = MIN(histEntryData.oID.oLength, histEntryData.oID.iLength);
  buf = malloc(bufLen + 1);
  memcpy(buf, histEntryData.oID.pioData, bufLen);
  buf[bufLen] = '\0';
  printf("    ID: %s\n", buf);
  free(buf);

  bufLen =
    MIN(histEntryData.oTableQualifier.oLength,
 histEntryData.oTableQualifier.iLength);
```

```
 buf = malloc(bufLen + 1);
 memcpy(buf, histEntryData.oTableQualifier.pioData, bufLen);
 buf[bufLen] = '\0';
 printf("    table qualifier: %s\n", buf);
 free(buf);

 bufLen =
   MIN(histEntryData.oTableName.oLength, histEntryData.oTableName.iLength);
 buf = malloc(bufLen + 1);
 memcpy(buf, histEntryData.oTableName.pioData, bufLen);
 buf[bufLen] = '\0';
 printf("    table name: %s\n", buf);
 free(buf);

 bufLen =
   MIN(histEntryData.oLocation.oLength, histEntryData.oLocation.iLength);
 buf = malloc(bufLen + 1);
 memcpy(buf, histEntryData.oLocation.pioData, bufLen);
 buf[bufLen] = '\0';
 printf("    location: %s\n", buf);
 free(buf);

 bufLen =
   MIN(histEntryData.oComment.oLength, histEntryData.oComment.iLength);
 buf = malloc(bufLen + 1);
 memcpy(buf, histEntryData.oComment.pioData, bufLen);
 buf[bufLen] = '\0';
 printf("    comment: %s\n", buf);
 free(buf);

 bufLen =
   MIN(histEntryData.oCommandText.oLength,
histEntryData.oCommandText.iLength);

 buf = malloc(bufLen + 1);
 memcpy(buf, histEntryData.oCommandText.pioData, bufLen);
 buf[bufLen] = '\0';
 printf("    command text: %s\n", buf);
 printf("    history file entry ID: %u\n", histEntryData.oEID.ioHID);
 printf("    table spaces:\n");
 free(buf);

 for (tsNb = 0; tsNb < histEntryData.oNumTablespaces; tsNb++)
 {
   bufLen =
     MIN(histEntryData.poTablespace[tsNb].oLength,
  histEntryData.poTablespace[tsNb].iLength);
   buf = malloc(bufLen + 1);
   memcpy(buf, histEntryData.poTablespace[tsNb].pioData, bufLen);
   buf[bufLen] = '\0';
   printf("      %s\n", buf);
   free(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);

 bufLen = SQLUDF_SQLSTATE_LEN;
 buf = malloc(bufLen + 1);
 memcpy(buf, histEntryData.poEventSQLCA->sqlstate, bufLen);
 buf[bufLen] = '\0';
 printf("      sqlstate: %s\n", buf);
```

```
  free(buf);

  bufLen = histEntryData.poEventSQLCA->sqlerrml;
  buf = malloc(bufLen + 1);
  memcpy(buf, histEntryData.poEventSQLCA->sqlerrmc, bufLen);
  buf[bufLen] = '\0';
  printf("      message: %s\n", buf);
  free(buf);

  return 0;
} /* HistoryEntryDisplay */

/******************************************************************************/
/* HistoryEntryDataFieldsFree                */
/* Deallocates the memory for database recovery history file structures    */
/******************************************************************************/
int HistoryEntryDataFieldsFree(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb = 0;

  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);
  pHistEntryData->oObjectPart.pioData = NULL;
  pHistEntryData->oEndTime.pioData = NULL;
  pHistEntryData->oFirstLog.pioData = NULL;
  pHistEntryData->oLastLog.pioData = NULL;
  pHistEntryData->oID.pioData = NULL;
  pHistEntryData->oTableQualifier.pioData = NULL;
  pHistEntryData->oTableName.pioData = NULL;
  pHistEntryData->oLocation.pioData = NULL;
  pHistEntryData->oComment.pioData = NULL;
  pHistEntryData->oCommandText.pioData = NULL;
  pHistEntryData->poEventSQLCA = NULL;

  for (tsNb = 0; tsNb < 3; tsNb++)
  {
    free(pHistEntryData->poTablespace[tsNb].pioData);
    pHistEntryData->poTablespace[tsNb].pioData = NULL;
  }

  free(pHistEntryData->poTablespace);
  pHistEntryData->poTablespace = NULL;

  return 0;
} /* HistoryEntryDataFieldsFree */
```

dblogconn sample program:

The dblogconn sample files show how to read database log files with a database connection.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
```

```
** (C) COPYRIGHT International Business Machines Corp. 2003
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*****************************************************************************
**
** SOURCE FILE NAME: dblogconn.sqc
**
** SAMPLE: How to read database log files asynchronously with a database
**         connection
**
**         Note:
**           You must be initially disconnected from the sample database
**           to run this program. To ensure you are, enter 'db2 connect
**           reset' on the command line prior to running dblogconn.
**
** DB2 API USED:
**         db2CfgSet -- Set Configuration
**         db2ReadLog -- Asynchronous Read Log
**
** SQL STATEMENTS USED:
**         ALTER TABLE
**         COMMIT
**         DELETE
**         INSERT
**         ROLLBACK
**         CONNECT RESET
**
** OUTPUT FILE: dblogconn.out (available in the online documentation)
*****************************************************************************
**
** For detailed information about database backup and database 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 on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
*****************************************************************************/
#include "utilrecov.c"
#include "utilemb.h"

/* local function prototypes */
int DbLogRecordsForCurrentConnectionRead(char *, char *, char *, char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1] = { 0 };
  char serverWorkingPath[SQL_PATH_SZ + 1] = { 0 };
  sqluint16 savedLogRetainValue = 0;
  char dbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char user[USERID_SZ + 1] = { 0 };
  char pswd[PSWD_SZ + 1] = { 0 };

  /* check the command line arguments */
```

```
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  CHECKRC(rc, "CmdLineArgsCheck3");

  printf("\nTHIS SAMPLE SHOWS HOW TO READ DATABASE LOGS ASYNCHRONOUSLY\n");
  printf("WITH A DATABASE CONNECTION.\n");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  CHECKRC(rc, "Instance Attach");

  /* get the server working path */
  rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
  CHECKRC(rc, "ServerWorkingPathGet");

  /* call the function to do asynchronous log read */
  rc = DbLogRecordsForCurrentConnectionRead(dbAlias,
                                            user, pswd, serverWorkingPath);
  CHECKRC(rc, "DbLogRecordsForCurrentConnectionRead");

  /* Detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  CHECKRC(rc, "InstanceDetach");

  return 0;
} /* end main */

int DbLogRecordsForCurrentConnectionRead(char dbAlias[],
                                         char user[],
                                         char pswd[],
                                         char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain = 0;

  db2BackupStruct backupStruct;
  db2TablespaceStruct tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32 backupImageSize = 0;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  SQLU_LSN startLSN;
  SQLU_LSN endLSN;
  char *logBuffer = NULL;
  sqluint32 logBufferSize = 0;
  db2ReadLogInfoStruct readLogInfo;
  db2ReadLogStruct readLogInput;
  int i = 0;

  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");

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

/* Calling up the routine for database backup */
rc = DbBackup(dbAlias, user, pswd, serverWorkingPath, &backupStruct);
CHECKRC(rc, "DbBackup");

/* connect to the database */
rc = DbConn(dbAlias, user, pswd);
CHECKRC(rc, "DbConn");

/* 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;

db2ReadLog(db2Version810, &readLogInput, &sqlca);
DB2_API_CHECK("database log info -- get");

logBufferSize = 64 * 1024;    /* Maximum size of a log buffer */
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;

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);
  CHECKRC(rc, "LogBufferDisplay");

  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.
     */
    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);
    CHECKRC(rc, "LogBufferDisplay");
  }

  /* free the log buffer */
  free(logBuffer);
  logBuffer = NULL;
  logBufferSize = 0;

  /* disconnect from the database */
  rc = DbDisconn(dbAlias);
  CHECKRC(rc, "DbDisconn");

  return 0;
} /* DbLogRecordsForCurrentConnectionRead */
```

dblognoconn sample program:

The dblognoconn sample files show how to read database log files with no database connection.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 2003
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
****************************************************************************
**
** SOURCE FILE NAME: dblognoconn.sqc
**
```

```
** SAMPLE: How to read database log files asynchronously
**         with no database connection
**
**         This program ends in ".sqc" even though it does not contain
**         embedded SQL statements. It links in  the embedded SQL utility
**         file for database connection and disconnection, so it needs the
**         embedded SQL extension for the precompiler.
**
**         Note:
**           You must be disconnected from the sample database to run
**           this program. To ensure you are, enter 'db2 connect reset'
**           on the command line prior to running dblognoconn.
**
** DB2 API USED:
**         db2CfgSet -- Set Configuration
**         db2ReadLogNoConnInit -- Read log without a db connection
**         db2ReadLog -- Asynchronous Read Log
**
** OUTPUT FILE: dblognoconn.out (available in the online documentation)
******************************************************************************
**
** For detailed information about database backup and database 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 on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
******************************************************************************/
#include "utilrecov.c"
#include "utilemb.h"

/* local function prototypes */
int DbReadLogRecordsNoConn(char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1] = { 0 };
  char serverWorkingPath[SQL_PATH_SZ + 1] = { 0 };
  sqluint16 savedLogRetainValue = 0;
  char dbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char user[USERID_SZ + 1] = { 0 };
  char pswd[PSWD_SZ + 1] = { 0 };

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  CHECKRC(rc, "CmdLineArgsCheck3");

  printf("\nTHIS SAMPLE SHOWS HOW TO READ DATABASE LOGS ASYNCHRONOUSLY\n");
  printf("WITH NO DATABASE CONNECTION.\n");

  /* get the server working path */
  rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
  CHECKRC(rc, "ServerWorkingPathGet");

  rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);
```

```
  CHECKRC(rc, "DbRecoveryHistoryFilePrune");

  /* Detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  CHECKRC(rc, "InstanceDetach");

  rc = DbReadLogRecordsNoConn(dbAlias);
  CHECKRC(rc, "DbReadLogRecordsNoConn");

  return 0;
} /* end main */

int DbReadLogRecordsNoConn(char dbAlias[])
{
  int rc = 0;
  struct sqlca sqlca;
  char logPath[SQL_PATH_SZ + 1] = { 0 };
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  char nodeName[] = "NODE0000\0";
  db2Uint32 readLogMemSize = 0;
  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);
  cfgParameters[0].ptrvalue = NULL;

  /*
   * First we must allocate memory for the API's control blocks and log
   * buffer
   */
  readLogMemSize = 4 * 4096;
  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 = DB2READLOG_RETRIEVE_OFF;
readLogInit.piDatabaseName = dbAlias;
readLogInit.piNodeName = nodeName;
readLogInit.iReadLogMemoryLimit = readLogMemSize;
readLogInit.poReadLogMemPtr = &readLogMemory;

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;

db2ReadLogNoConn(db2Version810, &readLogInput, &sqlca);
if (sqlca.sqlcode != 0)
{
  DB2_API_CHECK("database logs no db conn -- query");
}
/* Read some log records */
logBufferSize = 64 * 1024; /* Maximum size of a log buffer */
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;

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);
CHECKRC(rc, "LogBufferDisplay");

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.
     */
    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);
    CHECKRC(rc, "LogBufferDisplay");
  }

  printf("\nRead to end of logs.\n\n");
  free(logBuffer);
  logBuffer = NULL;
  logBufferSize = 0;

  readLogTerm.poReadLogMemPtr = &readLogMemory;

  db2ReadLogNoConnTerm(db2Version810, &readLogTerm, &sqlca);
  DB2_API_CHECK("database logs no db conn -- terminate");

  return 0;
} /* DbReadLogRecordsNoConn */
```

dbrestore sample program:

The dbrestore sample files show how to restore a database from a backup image.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 2003
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
****************************************************************************
**
** SOURCE FILE NAME: dbrestore.sqc
**
** SAMPLE: How to restore a database from a backup
**
**         This program ends in ".sqc" even though it does not contain
**         embedded SQL statements. It links in  the embedded SQL utility
**         file for database connection and disconnection, so it needs the
**         embedded SQL extension for the precompiler.
**
**         Note:
**           You must be disconnected from the sample database to run
**           this program. To ensure you are, enter 'db2 connect reset'
**           on the command line prior to running dbrestore.
**
** DB2 API USED:
**         db2CfgSet -- Set Configuration
**         db2Restore -- Restore Database
**
** OUTPUT FILE: dbrestore.out (available in the online documentation)
****************************************************************************
**
** For detailed information about database backup and database 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 on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
******************************************************************************/
#include "utilrecov.c"

/* local function prototypes */
int DbBackupAndRestore(char *, char *, char *, char *, char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1] = { 0 };
  char serverWorkingPath[SQL_PATH_SZ + 1] = { 0 };
  char restoredDbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char dbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char user[USERID_SZ + 1] = { 0 };
  char pswd[PSWD_SZ + 1] = { 0 };

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  CHECKRC(rc, "CmdLineArgsCheck3");

  printf("\nTHIS SAMPLE SHOWS HOW TO RESTORE A DATABASE FROM A\n");
  printf("BACKUP.\n");

  strcpy(restoredDbAlias, dbAlias);

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  CHECKRC(rc, "Instance Attach");

  /* get the server working path */
  rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
  CHECKRC(rc, "ServerWorkingPathGet");

  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");

  /* prune the recovery history file */
  rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);
  CHECKRC(rc, "DbRecoveryHistoryFilePrune");

  rc = DbBackupAndRestore(dbAlias,
                          restoredDbAlias, user, pswd, serverWorkingPath);
  CHECKRC(rc, "DbBackupAndRestore");

  /* Detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  CHECKRC(rc, "InstanceDetach");

  return 0;
} /* end main */
```

```
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 = 0;
  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1] = { 0 };

  db2BackupStruct backupStruct;
  db2TablespaceStruct tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32 backupImageSize = 0;
  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");

  /*****************************/
  /*    BACKUP THE DATABASE    */
  /*****************************/

  /* Calling up the routine for database backup */
  rc = DbBackup(dbAlias, user, pswd, serverWorkingPath, &backupStruct);
  CHECKRC(rc, "DbBackup");

  /*****************************/
  /*    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 = 2;
  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 */
```

dbrollfwd sample program:

The dbrollfwd sample files show how to perform rollforward operation after a database restore operation.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
```

```
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 2003
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
******************************************************************************
**
** SOURCE FILE NAME: dbrollfwd.sqc
**
** SAMPLE: How to perform rollforward after restore of a database
**
**         This program ends in ".sqc" even though it does not contain
**         embedded SQL statements. It links in  the embedded SQL utility
**         file for database connection and disconnection, so it needs the
**         embedded SQL extension for the precompiler.
**
**         Note:
**           You must be disconnected from the sample database to run
**           this program. To ensure you are, enter 'db2 connect reset'
**           on the command line prior to running dbrollfwd.
**
** DB2 APIs USED:
**         db2CfgSet -- Set Configuration
**         db2Restore -- Restore Database
**         db2Rollforward -- Rollforward Database
**
** OUTPUT FILE: dbrollfwd.out (available in the online documentation)
******************************************************************************
**
** For detailed information about database backup and database 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 on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
******************************************************************************/
#include "utilrecov.c"

/* local function prototypes */
int DbBackupRestoreAndRollforward(char *, char *, char *, char *, char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1] = { 0 };
  char serverWorkingPath[SQL_PATH_SZ + 1] = { 0 };
  char rolledForwardDbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char dbAlias[SQL_ALIAS_SZ + 1] = { 0 };
  char user[USERID_SZ + 1] = { 0 };
  char pswd[PSWD_SZ + 1] = { 0 };

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  CHECKRC(rc, "CmdLineArgsCheck3");
```

```
  printf("\nTHIS SAMPLE SHOWS HOW TO PERFORM ROLLFORWARD AFTER\n");
  printf("RESTORE OF A DATABASE.\n");
  strcpy(rolledForwardDbAlias, "RFDB");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  CHECKRC(rc, "Instance Attach");

  /* get the server working path */
  rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
  CHECKRC(rc, "ServerWorkingPathGet");

  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");

  rc = DbBackupRestoreAndRollforward(dbAlias, rolledForwardDbAlias, user,
                                     pswd, serverWorkingPath);
  CHECKRC(rc, "DbBackupRestoreAndRollforward");

  /* Detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  CHECKRC(rc, "InstanceDetach");

  return 0;
} /* end main */

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 = 0;
  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1] = { 0 };
  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] = { 0 };
  sqlint32 numReplies = 0;
  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 ROLLFORWARD A DATABASE.\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");

/*****************************/
/*    BACKUP THE DATABASE    */
/*****************************/

/* Calling the routine for database backup */
rc = DbBackup(dbAlias, user, pswd, serverWorkingPath, &backupStruct);
CHECKRC(rc, "DbBackup");

/* 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);
CHECKRC(rc, "DbCreate");
*/

/*****************************/
/*    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 = 2;
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 forward database '%s'...\n", rolledForwardDbAlias);
  rfwdInput.iVersion = SQLUM_RFWD_VERSION;
  rfwdInput.piDbAlias = rolledForwardDbAlias;
  rfwdInput.iCallerAction = DB2ROLLFORWARD_RFWD_STOP;
  rfwdInput.piStopTime = SQLUM_INFINITY_TIMESTAMP;
  rfwdInput.piUserName = user;
  rfwdInput.piPassword = pswd;
  rfwdInput.piOverflowLogPath = serverWorkingPath;
  rfwdInput.iNumChngLgOvrflw = 0;
  rfwdInput.piChngLogOvrflw = NULL;
  rfwdInput.iConnectMode = DB2ROLLFORWARD_OFFLINE;
  rfwdInput.piTablespaceList = NULL;
  rfwdInput.iAllNodeFlag = DB2_ALL_NODES;
  rfwdInput.iNumNodes = 0;
  rfwdInput.piNodeList = NULL;
  rfwdInput.piDroppedTblID = NULL;
  rfwdInput.piExportDir = NULL;
  rfwdInput.iNumNodeInfo = 1;
  rfwdInput.iRollforwardFlags = DB2ROLLFORWARD_EMPTY_FLAG;
  rfwdOutput.poApplicationId = rollforwardAppId;
  rfwdOutput.poNumReplies = &numReplies;
  rfwdOutput.poNodeInfo = &nodeInfo;
  rfwdStruct.piRfwdInput = &rfwdInput;
  rfwdStruct.poRfwdOutput = &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);
  CHECKRC(rc, "DbDrop");
  return 0;
} /* DbBackupRestoreAndRollforward */
```

Appendix F. Cross-node recovery with the db2adutl command and the logarchopt1 and vendoropt database configuration parameters

The examples that follow show how to perform cross-node recovery using the db2adutl command, and the *logarchopt1* and *vendoropt* database configuration parameters.

For the following examples, computer 1 is called `bar` and is running AIX. The owner of this machine is `roecken`. The database on `bar` is called `zample`. Computer 2 is called `dps`. This machine is also running AIX, and is owned by `regress9`.

PASSWORDACCESS = generate:

Computer 1:

1. Set up the database for log archiving to TSM. Update the database configuration parameter *logarchmeth1* for the `zample` database:

   ```
   bar:/home/roecken> db2 update db cfg for zample using LOGARCHMETH1 tsm
   ```

   The following information is returned:

   ```
   DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
   ```

   Note: Before updating the database configuration, you might have to take an offline backup of the database.

2. Force off applications:

   ```
   db2 force applications all
   ```

3. Verify that all applications have been forced off:

   ```
   db2 list applications
   ```

   You should receive a message that says no data was returned.

   Note: In a partitioned database environment, you must perform this step on all database partitions.

4. Take a backup of the database:

   ```
   db2 backup db zample use tsm
   ```

   Information similar to the following is returned:

   ```
   Backup successful. The timestamp for this backup image is : 20040216151025
   ```

   Note: In a partitioned database environment, you must perform this step on all database partitions. You must backup the catalog partition first, then you can back up all other database partitions concurrently.

5. Connect to the `zample` database, then create a table in it.
6. Load data into the new table. In this example, the table is called `a`, and the data is being loaded from a delimited ASCII file called `mr`. The COPY YES option is specified to make a copy of the data that is loaded, and the USE TSM option specifies that the copy of the data is stored on Tivoli Storage Manager.

   Note: You can only specify the COPY YES option if the database is enabled for rollforward recovery; that is, the *logarchmeth1* database configuration parameter must be set to either USEREXIT or LOGRETAIN.

```
bar:/home/roecken> db2 load from mr of del modified by noheader replace
   into a copy yes use tsm
```

The utility returns a series of messages to indicate its progress:

```
SQL3109N  The utility is beginning to load data from file "/home/roecken/mr".

SQL3500W  The utility is beginning the "LOAD" phase at time "02/16/2004
15:12:13.392633".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3110N  The utility has completed processing.  "1" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "1".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "02/16/2004
15:12:13.445718".


Number of rows read         = 1
Number of rows skipped      = 0
Number of rows loaded       = 1
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1
```

There should now be one backup image, one load copy image and one log file on TSM. A query on the `zample` database can be run as follows:

```
bar:/home/roecken/sqllib/adsm> db2adutl query db zample
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 0
    Sessions: 1


Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE


Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE


Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE


Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE


Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE


Retrieving LOAD COPY information.
    1 Time: 20040216151213
```

```
Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

7. To enable cross-node recovery, another node and account must be given access to the objects on the bar computer. In this example, access is given to the node dps and the user regress9.

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl grant user regress9
      on nodename dps for db zample
   ```

   The following information is returned:

   ```
   Successfully added permissions for regress9 to access ZAMPLE on node dps.
   ```

   To query the results of the **db2adutl grant** operation, issue the following command:

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl queryaccess
   ```

   The following information is returned:

   ```
   Node                 Username             Database Name   Type
   ------------------------------------------------------------
   DPS                  regress9             ZAMPLE          A
   ------------------------------------------------------------
    Access Types:  B - backup images   L - logs   A - both
   ```

PASSWORDACCESS = generate environment:

Computer 2:

Computer 2, dps, is not yet set up. A **db2adutl** query on dps for the zample database returns the following results:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample
--- Database directory is empty ---
Warning: There are no file spaces created by DB2 on the ADSM server
Warning: No DB2 backup images found in ADSM for any alias.


dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename
    bar owner roecken
--- Database directory is empty ---

Query for database ZAMPLE


Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 0
    Sessions: 1


Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE


Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE


Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE


Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE


Retrieving LOAD COPY information.
    1 Time: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

The zample database does not yet exist on the dps computer.

1. Restore the zample database to the dps computer:

   ```
   dps:/home/regress9> db2 restore db zample use tsm options
   "'-fromnode=bar -fromowner=roecken'" without prompting
   ```

   The following information is returned:

   ```
   DB20000I  The RESTORE DATABASE command completed successfully.
   ```

   Note: If the zample database already existed on dps, the OPTIONS parameter would be omitted, and the database configuration parameter *vendoropt* would be used. This configuration parameter overrides the OPTIONS parameter for a backup or restore operation.

   A rollforward operation on the zample database will fail because the rollforward utility cannot find the log files. A rollforward operation such as the following:

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   Returns the following error:

   ```
   SQL4970N  Roll-forward recovery on database "ZAMPLE" cannot reach the
   specified stop point (end-of-log or point-in-time) because of missing log
   file(s) on node(s) "0".
   ```

2. To force the rollforward utility to look for log files on another machine, you must configure the proper *logarchopt* value, in this situation the *logarchopt1* database configuration parameter:

   ```
   dps:/home/regress9> db2 update db cfg for zample using logarchopt1
   "'-fromnode=bar -fromowner=roecken'"
   ```

3. For the rollforward utility to be able to use the load copy images, you must also set the *vendoropt* database configuration parameter:

   ```
   dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
   "'-fromnode=bar -fromowner=roecken'"
   ```

4. The zample database can now be rolled forward::

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   The following information is returned:

   ```
                                  Rollforward Status

   Input database alias                   = zample
   Number of nodes have returned status   = 1

   Node number                            = 0
   Rollforward status                     = not pending
   Next log file to be read               =
   Log files processed                    = S0000000.LOG - S0000000.LOG
   Last committed transaction             = 2004-02-16-20.10.38.000000 UTC

  DB20000I  The ROLLFORWARD command completed successfully.
   ```

PASSWORDACCESS = prompt environment:

In a PROMPT environment, extra information is required, specifically the TSM nodename and password of the machine where the objects were created.

For **db2adutl**, update the dsm.sys file (called the *dsm.opt* file on Windows-based platforms) and add NODENAME bar (because bar is the name of the source computer) to the server clause:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename bar
owner roecken password *******
```

The following information is returned:

```
Query for database ZAMPLE


Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 0
    Sessions: 1


Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE


Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE


Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE


Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE


Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE


Retrieving LOAD COPY information.
    1 Time: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

1. If the database does not exist, create an empty zample database. If the zample database already exists, this step, and the next two steps that update the database configuration, can be skipped.

   ```
   dps:/home/regress9> db2 create db zample
   ```

2. Update the database configuration parameter *tsm_nodename* for the zample database:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
   ```

3. Update the database configuration parameter *tsm_password* for the zample database:

   ```
   dps:/home/regress9> db2 update db cfg for zample using
   tsm_password ********
   ```

4. Restore the zample database:

   ```
   dps:/home/regress9> db2 restore db zample use tsm options
   "'-fromnode=bar -fromowner=roecken'" without prompting
   ```

   The restore operation completes successfully, but a warning is issued:

```
SQL2540W  Restore is successful, however a warning "2523" was
encountered during Database Restore while processing in No
Interrupt mode.
```

Again, at this point, the rollforward utility cannot find the correct log files:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following error message is returned:

```
SQL1268N  Roll-forward recovery stopped due to error "-2112880618"
while retrieving log file "S0000000.LOG" for database "ZAMPLE" on node "0".
```

5. Because the database restore operation replaces the database configuration file, the TSM database configuration values must be set to the correct values. First the *tsm_nodename* configuration parameter must be reset:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
   ```

6. The *tsm_password* database configuration parameter must be reset:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_password *******
   ```

7. The *logarchopt1* database configuration parameter must be reset so the rollforward utility can find the correct log files:

   ```
   dps:/home/regress9> db2 update db cfg for zample using logarchopt1
   "'-fromnode=bar -fromowner=roecken'"
   ```

8. The *vendoropt* database configuration parameter must also be reset so that the load recovery file can also be used:

   ```
   dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
   "'-fromnode=bar -fromowner=roecken'"
   ```

9. When the database configuration parameters are set, the database can be rolled forward:

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   A ROLLFORWARD QUERY STATUS command on the `zample` database shows the following:

```
                                 Rollforward Status

 Input database alias                   = zample
 Number of nodes have returned status   = 1

 Node number                            = 0
 Rollforward status                     = not pending
 Next log file to be read               =
 Log files processed                    = S0000000.LOG - S0000000.LOG
 Last committed transaction             = 2004-02-16-20.10.38.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

Related reference:

- "db2adutl - Managing DB2 objects within TSM" on page 303
- "logarchopt1 - Primary log archive options configuration parameter" in *Performance Guide*
- "vendoropt - Vendor options configuration parameter" in *Performance Guide*

Appendix G. 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 the following:

- 64-bit Solaris systems, which require TSM client API Version 4.2.1.
- 64-bit Windows operating systems, which require TSM client API Version 5.1.
- All Windows X64 systems, which require TSM client API Version 5.3.2.
- 32-bit Linux for iSeries and pSeries®, which require at minimum TSM client API Version 5.1.5
- 64-bit Linux for iSeries and pSeries, which require at minimum TSM client API Version 5.2.2
- 64-bit Linux on AMD Opteron systems, which require a minimum TSM client API Version 5.2.0.
- Linux for zSeries®, which requires a minimum TSM client API Version 5.2.2.

Configuring a Tivoli Storage Manager client

Before the database manager can use the TSM option, the following steps might 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 on each DB2 server.
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 might 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 might 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.
- TSM does not allow identical object names. If two different databases are backed up to TSM and they have the same database alias name, node number, and timestamp of backup, TSM marks one of them as inactive. You should use a unique TSM database alias names for each DB2 instance.
- 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 might 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 might be too large.
  - Database activity during an online backup operation might be too high.
- Use multiple sessions to increase throughput (only if sufficient hardware is available on the TSM server).
- If you perform an online backup operation and specify the USE TSM option and the INCLUDE LOGS option, a deadlock can occur if the two processes try to write to the same tape drive at the same time. If you are using a tape drive as a storage device for logs and backup images, you need to define two separate tape pools for TSM, one for the backup image and one for the archived logs.

Related concepts:

- "Log file management" on page 46

Related reference:

- "db2adutl - Managing DB2 objects within TSM" on page 303

Appendix H. Tivoli Space Manager Hierarchical Storage Management support for partitioned tables

The Tivoli Space Manager Hierarchical Storage Manager (HSM) client program automatically migrates eligible files to secondary storage to maintain specific levels of free space on local file systems.

With table partitioning, table data is divided across multiple storage objects called data partitions. HSM supports the backup of individual data partitions to secondary storage.

When using SMS table spaces, each data partition range is represented as a file in the corresponding directory. Therefore, it is very easy to migrate individual ranges of data (data partitions) to secondary storage.

When using DMS table spaces, each container is represented as a file. In this case, infrequently accessed ranges should be stored in their own table space. When you issue a CREATE TABLE statement using the EVERY clause, use the NO CYCLE clause to ensure that the number of table spaces listed in the table level IN clause match the number of data partitions being created. This is demonstrated in the following example:

Example 1

```
CREATE TABLE t1 (c INT) IN tbsp1, tbsp2, tbsp3 NO CYCLE
  PARTITION BY RANGE(c)
  (STARTING FROM 2 ENDING AT 6 EVERY 2);
```

Related concepts:

- "Log file management" on page 46

Related reference:

- "CREATE TABLE statement" in *SQL Reference, Volume 2*
- "db2adutl - Managing DB2 objects within TSM" on page 303
- Appendix G, "Tivoli Storage Manager," on page 403

Appendix I. 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 *logarchmeth1* database configuration parameter has been set to `USEREXIT`. 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 411). 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 might be asked to perform.

The following topics are covered:

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 database.

- **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 9 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 9. 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]
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]

Table 9. User Exit Program Return Codes (continued). Applies to archiving and retrieval operations only.

Return Code	Explanation
[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 might not have been successfully archived previously. If the user exit program fails to archive log files, your disk might become filled with log files, and performance might 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.	

Related concepts:

- "Log file management through log archiving" on page 49

Related reference:

- "Configuration parameters for database logging" on page 37

Appendix J. Backup and restore APIs for vendor products

DB2 APIs for backup and restore to storage managers

DB2 provides an interface that can be used by third-party media management products to store and retrieve data for backup and restore operations and log files. This interface is designed to augment the backup, restore, and log archiving 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 section.

DB2 defines a set of API prototypes that provide a general purpose data interface to backup, restore, and log archiving that can be used by many vendors. These APIs are to be provided by the vendor in a shared library on UNIX based systems, or DLL on the Windows operating system. When the APIs are invoked by DB2, the shared library or DLL specified by the calling backup, restore, or log archiving routine is loaded and the APIs provided by the vendor are called to perform the required tasks.

Sample files demonstrating the DB2 vendor functionality are located on UNIX platforms in the `sqllib/samples/BARVendor` directory, and on Windows in the `sqllib\samples\BARVendor` directory.

The following are the definitions for terminology used in the descriptions of the backup and restore vendor storage plug-in APIs.

Backup and restore vendor storage plug-in
A dynamically loadable library that DB2 will load to access user-written backup and restore APIs for vendor products.

Input Indicates that DB2 will fill in the value for the backup and restore vendor storage plug-in API parameter.

Output
Indicates that the backup and restore vendor storage plug-in API will fill in the value for the API parameter.

Operational overview

Seven APIs are defined to provide a data interface between DB2 and the vendor product:

- sqluvint - Initialize and link to a vendor device
- sqluvget - Read data from a vendor device
- sqluvput - Write data to a vendor device
- sqluvend - Unlink the device and release its resources
- sqluvdel - Delete committed session
- db2VendorQueryApiVersion - Query device supported API level
- db2VendorGetNextObj - Get next object on device

DB2 will call these APIs, 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 API may compromise data integrity of the database.

The sequence of APIs 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, a restore, a log archive, or a log retrieve operation.
- The PROMPTING mode that is specified on the backup or restore operation.
- The characteristics of the vendor 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 APIs 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 APIs.

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 the sqluvint API.

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 backing up an image to a vendor device, the following sequence of calls is issued by DB2 for *each* session:

```
db2VendorQueryApiVersion
```

followed by 1

```
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.

Note: For backing up a log file to a vendor device, use action = SQLUV_ARCHIVE with the sqluvint call.

For restoring an image from a vendor device, the sequence of calls for each session is:

```
db2VendorQueryApiVersion
```

followed by 1

```
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: For restoring a log file from a vendor device, use action = SQLUV_RETRIEVE with the sqluvint call.

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.

For searching a vendor device for an image or archived log file, the following sequence of calls is issued by DB2 for each session:

```
sqluvint, action = SQLUV_QUERY_IMAGES or SQLUV_QUERY_LOGS
```

followed by 1 to n

```
db2VendorGetNextObj
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

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 restore operation completes.
- If the response is *terminate*, DB2 terminates the backup or restore operation. For more information on exactly what DB2 does to terminate the sessions, see "If error conditions are returned to DB2."

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 and the LOAD keyword was used, the DEVICE field in the history record contains an `0`. If the backup operation was directed to TSM, the DEVICE field contains an `A`. 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	*options-string*	*options-string*
Vendor file name	*file-name*	*file-name*
Transfer buffer size	*buffer-size*	*buffer-size*

Backup and restore API function calls

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 concepts:

- "DB2 database system plug-ins for customizing database management" in *Administrative API Reference*

Related reference:

- "db2VendorGetNextObj - Get next object on device" on page 432
- "db2VendorQueryApiVersion - Get the supported level of the vendor storage API" on page 431
- "Data " on page 440
- "DB2_info " on page 434
- "Init_input " on page 438
- "Init_output " on page 439
- "Return_code " on page 440
- "sqluvdel - Delete committed session" on page 430
- "sqluvend - Unlink a vendor device and release its resources" on page 429

- "sqluvget - Read data from a vendor device" on page 426
- "sqluvint - Initialize and link to a vendor device" on page 421
- "sqluvput - Write data to a vendor device" on page 427
- "Vendor_info " on page 437

sqluvint - Initialize and link to a vendor device

Provides information for initializing a vendor device and for establishing a logical link between DB2 and the vendor device.

Authorization:

None

Required connection:

Database

API include file:

```
sqluvend.h
```

API and data structure syntax:

```
int sqluvint ( struct Init_input *in,
               struct Init_output *out,
               struct Return_code *return_code);
```

sqluvint API parameters:

in Input. Structure that contains information provided by DB2 to establish a logical link with the vendor device.

out 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 API to obtain a device handle. If for any reason, the vendor storage API 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 API. Details on possible return codes, and the DB2 reaction to each of these, is contained in the return codes table (see table below).

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 calling other data transfer APIs, but may terminate the session at any time with an sqluvend call.

Return codes:

Table 10. 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 to the sqluvput API (to BACKUP data). If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_OK; the next call will be to the sqluvget API 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 API call will not be received, since the session was never established.

Table 10. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_COMM_ ERROR	Communication error with device.	No further calls.	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend API 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 API 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 API 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 API 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 API call will not be received, since the session was never established.

Table 10. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
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 API 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 API 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 API call will not be received, since the session was never established.
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 API 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 API call will not be received, since the session was never established.

Table 10. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_DEV_ ERROR	Device error.	No further calls.	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend API 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 to sqluvput API. If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_MAX_LINK_ GRANT; the next call will be to the sqluvget API 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 API 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 API call will not be received, since the session was never established.

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413
- "Init_input " on page 438
- "Init_output " on page 439
- "Return_code " on page 440

sqluvget - Read data from a vendor device

After a vendor device has been initialized with the sqluvint API, DB2 calls this API to read from the device during a restore operation.

Authorization:

None

Required connection:

Database

API include file:

```
sqluvend.h
```

API and data structure syntax:

```
int sqluvget ( void * hdle,
               struct Data *data,
               struct Return_code *return_code);
```

sqluvget API parameters:

hdle Input. Pointer to space allocated for the Data structure (including the data buffer) and Return_code.

data Input or output. A pointer to the Data structure.

return_code
Output. The return code from the API call.

Usage notes:

This API is used by the restore utility.

Return codes:

Table 11. 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 (see note below	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT (see note below	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT (see note below	The session will be terminated.

Table 11. Valid Return Codes for sqluvget and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT (see note below	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT (see note below	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 (see note below	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 (see note below	The session will be terminated.

Note: Next call: If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated.

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413
- "Data " on page 440
- "Return_code " on page 440

sqluvput - Write data to a vendor device

After a vendor device has been initialized with the sqluvint API, DB2 calls this API to write to the device during a backup operation.

Authorization:

None

Required connection:

Database

API include file:

```
sqluvend.h
```

API and data structure syntax:

```
int sqluvput ( void *               hdle,
               struct Data *data,
               struct Return_code *return_code);
```

sqluvput API parameters:

hdle 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.

Usage notes:

This API is used by the backup utility.

Return codes:

Table 12. 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 (see note below).	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT (see note below).	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT (see note below).	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT (see note below).	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 (see note below).	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 (see note below).	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT (see note below).	The session will be terminated.

Note: Next call: 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.

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413
- "Data " on page 440

• "Return_code " on page 440

sqluvend - Unlink a vendor device and release its resources

Unlinks a vendor device and frees all of its related resources. All unused resources (for example, allocated space and file handles) must be released before the sqluvend API call returns to DB2.

Authorization:

None

Required connection:

Database

API include file:

```
sqluvend.h
```

API and data structure syntax:

```
int sqluvend ( sqlint32             action,
               void *hdle,
               struct Init_output *in_out,
               struct Return_code *return_code);
```

sqluvend API parameters:

action Input. Used to commit or abort the session:
- SQLUV_COMMIT (0 = to commit)
- SQLUV_ABORT (1 = to abort)

hdle Input. Pointer to the Init_output structure.

in_out 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 API 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 13. 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	

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413
- "Init_output " on page 439
- "Return_code " on page 440

sqluvdel - Delete committed session

Deletes committed sessions from a vendor device.

Authorization:

None

Required connection:

Database

API include file:

```
sqluvend.h
```

API and data structure syntax:

```
int sqluvdel ( struct Init_input *in,
               struct Init_output *vendorDevData,
               struct Return_code *return_code);
```

sqluvdel API parameters:

in Input. Space allocated for Init_input and Return_code.

vendorDevData
Output. Structure containing the output returned by the vendor device.

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 API 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 the sqluvdel API, an initial fatal error must have occurred.

Return codes:

Table 14. Valid return codes for sqluvdel and resulting database server action

Literal in header file	Description	Probable next call
SQLUV_OK	Operation successful	No further calls
SQLUV_DELETE_FAILED	Delete request failed	No further calls

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413
- "Init_input " on page 438
- "Init_output " on page 439
- "Return_code " on page 440

db2VendorQueryApiVersion - Get the supported level of the vendor storage API

Determines which level of the vendor storage API is supported by the backup and restore vendor storage plug-in. If the specified vendor storage plug-in is not compatible with DB2, then it will not be used.

If a vendor storage plug-in does not have this API implemented for logs, then it cannot be used and DB2 will report an error. This will not affect images that currently work with existing vendor libraries.

Authorization:

None

Required connection:

Database.

API include file:

```
db2VendorApi.h
```

API and data structure syntax:

```
void db2VendorQueryApiVersion ( db2Uint32  * supportedVersion );
```

db2VendorQueryApiVersion API parameters:

supportedVersion
Output. Returns the version of the vendor storage API the vendor library supports.

Usage notes:

This API will be called before any other vendor storage APIs are invoked.

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

db2VendorGetNextObj - Get next object on device

This API is called after a query has been set up (using the sqluvint API) to get the next object (image or archived log file) that matches the search criteria. Only one search for either images or log files can be set up at one time.

Authorization:

None

Required connection:

Database.

API include file:

```
db2VendorApi.h
```

API and data structure syntax:

```
int db2VendorGetNextObj ( void                        * vendorCB,
                          struct db2VendorQueryInfo   * queryInfo,
                          struct Return_code          * returnCode);

typedef struct db2VendorQueryInfo
{
   db2Uint64 sizeEstimate;
```

```
   db2Uint32 type;
   SQL_PDB_NODE_TYPE dbPartitionNum;
   db2Uint16 sequenceNum;
   char db2Instance[SQL_INSTNAME_SZ + 1];
   char dbname[SQL_DBNAME_SZ + 1];
   char dbalias[SQL_ALIAS_SZ + 1];
   char timestamp[SQLU_TIME_STAMP_LEN + 1];
   char filename[DB2VENDOR_MAX_FILENAME_SZ + 1];
   char owner[DB2VENDOR_MAX_OWNER_SZ + 1];
   char mgmtClass[DB2VENDOR_MAX_MGMTCLASS_SZ + 1];
   char oldestLogfile[DB2_LOGFILE_NAME_LEN + 1];
} db2VendorQueryInfo;
```

db2VendorGetNextObj API parameters:

vendorCB
Input. Pointer to space allocated by the vendor library.

queryInfo
Output. Pointer to a db2VendorQueryInfo structure to be filled in by the vendor library.

returnCode
Output. The return code from the API call.

db2VendorQueryInfo data structure parameters:

sizeEstimate
Specifies the estimated size of the object.

type Specifies the image type if the object is a backup image.

dbPartitionNum
Specifies the number of the database partition that the object belongs to.

sequenceNum
Specifies the file extension for the backup image. Valid only if the object is a backup.

db2Instance
Specifies the name of the instance that the object belongs to.

dbname
Specifes the name of the database that the object belongs to.

dbalias
Specifies the alias of the database that the object belongs to.

timestamp
Specifies the time stamp used to identify the backup image. Valid only if the object is a backup image.

filename
Specifies the name of the object if the object is a load copy image or an archived log file.

owner Specifies the owner of the object.

mgmtClass
Specifies the management class the object was stored under (used by TSM).

oldestLogfile
Specifies the oldest log file stored with a backup image.

Usage notes:

Not all parameters will pertain to each object or each vendor. The mandatory parameters that need to be filled out are db2Instance, dbname, dbalias, timestamp (for images), filename (for logs and load copy images), owner, sequenceNum (for images) and dbPartitionNum. The remaining parameters will be left for the specific vendors to define. If a parameter does not pertain, then it should be initialized to "" for strings and 0 for numeric types.

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413
- "Return_code " on page 440

DB2_info

Contains information about the DB2 product and the database that is being backed up or restored. This structure is used to identify DB2 to the vendor device and to describe a particular session between DB2 and the vendor device. It is passed to the backup and restore vendor storage plug-in as part of the Init_input data structure.

Table 15. Fields in the DB2_info Structure.

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.

Table 15. Fields in the DB2_info Structure. (continued)

Field Name	Data Type	Description
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.
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	Reserved for future use.
max_bytes_per_txn	sqlint32	Specifies to the vendor in bytes, the transfer buffer size specified by the user.

Table 15. Fields in the DB2_info Structure. (continued)

Field Name	Data Type	Description
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.

Note: All char data type fields are null-terminated strings.

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.

API and data structure syntax:

```
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;
  sqlint32 max_bytes_per_txn;
  char *image_filename;
  void *reserve;
  char *nodename;
  char *password;
  char *owner;
  char *mcNameP;
  SQL_PDB_NODE_TYPE nodeNum;
} DB2_info ;
```

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

Vendor_info

Contains information, returned to DB2 as part of the Init_output structure, identifying the vendor and the version of the vendor device.

Table 16. Fields in the Vendor_info Structure.

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.
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 API is SQLUV_BUFF_SIZE, indicating that an invalid buffer size wasspecified.
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.

Note: All char data type fields are NULL-terminated strings.

API and data structure syntax:

Vendor_info

```
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;
```

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

Init_input

Contains information provided by DB2 to set up and to establish a logical link with a vendor device. This data structure is used by DB2 to send information to the backup and restore vendor storage plug-in through the sqluvint and sqluvdel APIs.

Table 17. Fields in the Init_input Structure.

Field Name	Data Type	Description
DB2_session	struct DB2_info	A description of the session from the perspective of DB2.
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.

Table 17. Fields in the Init_input Structure. (continued)

Field Name	Data Type	Description
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. This field is a NULL-terminated string.
num_sessions	unsigned short	Number of sessions requested by the user when a backup or a restore operation was invoked.

API and data structure syntax:

```
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;
```

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

Init_output

Contains a control block for the session and information returned by the backup and restore vendor storage plug-in to DB2. This data structure is used by the sqluvint and sqluvdel APIs.

Table 18. 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.

API and data structure syntax:

```
typedef struct Init_output
{
   struct Vendor_info * vendor_session;
   void               * pVendorCB;
   void               * reserve;
} Init_output ;
```

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

Data

Contains data transferred between DB2 and a vendor device. This structure is used by the sqluvput API when data is being written to the vendor device and by the sqluvget API when data is being read from the vendor device.

Table 19. 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.
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.

API and data structure syntax:

```
typedef struct Data
{
   sqlint32 obj_num;
   sqlint32 buff_size;
   sqlint32 actual_buff_size;
   void *dataptr;
   void *reserve;
} Data;
```

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

Return_code

Contains the return code for and a short explanation of the error being returned to DB2 by the backup and restore vendor storage plug-in. This data structure is used by all the vendor storage plug-in APIs.

Table 20. Fields in the Return_code Structure

Field Name	Data Type	Description
return_code (see note below)	sqlint32	Return code from the vendor API.
description	char	A short description of the return code.
reserve	void	Reserved for future use.

Note: 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.

API and data structure syntax:

```
typedef struct Return_code
{
        sqlint32 return_code;
        char description[SQLUV_COMMENT_LEN];
        void *reserve;
} Return_code;
```

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 413

APIs for compressed backups

DB2 APIs for using compression with backup and restore operations

DB2 provides APIs that can be used by third-party compression products to compress and decompress backup images. This interface is designed to augment or replace the compression library that is supported as a standard part of DB2. The compression plug-in interface can be used with the backup and restore DB2 APIs or the backup and restore plug-ins for vendor storage devices.

DB2 defines a set of API prototypes that provide a general purpose interface for compression and decompression that can be used by many vendors. These APIs are to be provided by the vendor in a shared library on Linux and UNIX systems, or DLL on the Windows operating system. When the APIs are invoked by DB2, the shared library or DLL specified by the calling backup or restore routine is loaded and the APIs provided by the vendor are called to perform the required tasks.

Operational overview

Eight APIs are defined to interface DB2 and the vendor product:

- InitCompression - Initialize the compression library
- GetSavedBlock - Get vendor block for backup image
- Compress - Compress a block of data
- GetMaxCompressedSize - Estimate largest possible buffer size
- TermCompression - Terminate the compression library
- InitDecompression - Initialize the decompression library
- Decompress - Decompress a block of data
- TermDecompression - Terminate the decompression library

DB2 will provide the definition for the COMPR_DB2INFO structure; the vendor will provide definitions for each of the other structures and APIs for using compression with backup and restore. The structures, prototypes, and constants are defined in the file `sqlucompr.h`, which is shipped with DB2.

DB2 will call these APIs, and they should be provided by the vendor product in a shared library on Linux and UNIX 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 might compromise data integrity of the database.

Sample calling sequence

For backup, the following sequence of calls is issued by DB2 for each session:

```
InitCompression
```

followed by 0 to 1

```
GetMaxCompressedSize
Compress
```

followed by 1

```
TermCompress
```

For restore, the sequence of calls for each session is:

```
InitDecompression
```

followed by 1 to n

```
Decompress
```

followed by 1

```
TermCompression
```

Compression plug-in interface return codes

The following are the return codes that the APIs might return. Except where specified, DB2 will terminate the backup or restore when any non-zero return code is returned.

```
SQLUV_OK
```

0

Operation succeeded

```
SQLUV_BUFFER_TOO_SMALL
```

100

Target buffer is too small. When indicated on backup, the tgtAct field shall indicate the estimated size required to compress the object. DB2 will retry the operation with a buffer at least as large as specified. When indicated on restore, the operation will fail.

```
SQLUV_PARTIAL_BUFFER
```

101

A buffer was partially compressed. When indicated on backup, the srcAct field shall indicate the actual amount of data actually compressed and the tgtAct field shall indicate the actual size of the compressed data. When indicated on restore, the operation will fail.

```
SQLUV_NO_MEMORY
```

102

Out of memory

```
SQLUV_EXCEPTION
```

103

A signal or exception was raised in the code.

`SQLUV_INTERNAL_ERROR`

104

An internal error was detected.

The difference between SQLUV_BUFFER_TOO_SMALL and SQLUV_PARTIAL_BUFFER is that when SQLUV_PARTIAL_BUFFER is returned, DB2 will consider the data in the output buffer to be valid.

Related concepts:

- "DB2 database system plug-ins for customizing database management" in *Administrative API Reference*

Related reference:

- "Compress API - Compress a block of data" in *Administrative API Reference*
- "COMPR_CB data structure" in *Administrative API Reference*
- "COMPR_DB2INFO data structure" in *Administrative API Reference*
- "COMPR_PIINFO data structure" in *Administrative API Reference*
- "db2Backup - Back up a database or table space" on page 76
- "db2Restore - Restore a database or table space" on page 115
- "Decompress API - Decompress a block of data" in *Administrative API Reference*
- "GetMaxCompressedSize API - Estimate largest possible buffer size" in *Administrative API Reference*
- "GetSavedBlock API - Get the vendor of block data for the backup image" in *Administrative API Reference*
- "InitCompression API - Initialize the compression library" in *Administrative API Reference*
- "TermDecompression API - Stop the decompression library" in *Administrative API Reference*
- "InitDecompression API - Initialize the decompression library" in *Administrative API Reference*
- "TermCompression API - Stop the compression library" in *Administrative API Reference*

DB2 APIs for using compression with backup and restore operations

Appendix K. DB2 Database technical information

Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF CD)
  - printed books
- Command line help
  - Command help
  - Message help
- Sample programs

IBM periodically makes documentation updates available. If you access the online version on the DB2 Information Center at ibm.com®, you do not need to install documentation updates because this version is kept up-to-date by IBM. If you have installed the DB2 Information Center, it is recommended that you install the documentation updates. Documentation updates allow you to update the information that you installed from the *DB2 Information Center CD* or downloaded from Passport Advantage as new information becomes available.

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and Redbooks™ online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how we can improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

Related concepts:

- "Features of the DB2 Information Center" in *Online DB2 Information Center*
- "Sample files" in *Samples Topics*

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*
- "Updating the DB2 Information Center installed on your computer or intranet server" on page 451

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 446

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. DB2 Version 9 manuals in PDF format can be downloaded from www.ibm.com/software/data/db2/udb/support/manualsv9.html.

Although the tables identify books available in print, the books might not be available in your country or region.

The information in these books is fundamental to all DB2 users; you will find this information useful whether you are a programmer, a database administrator, or someone who works with DB2 Connect or other DB2 products.

Table 21. DB2 technical information

Name	Form Number	Available in print
Administration Guide: Implementation	SC10-4221	Yes
Administration Guide: Planning	SC10-4223	Yes
Administrative API Reference	SC10-4231	Yes
Administrative SQL Routines and Views	SC10-4293	No
Call Level Interface Guide and Reference, Volume 1	SC10-4224	Yes
Call Level Interface Guide and Reference, Volume 2	SC10-4225	Yes
Command Reference	SC10-4226	No
Data Movement Utilities Guide and Reference	SC10-4227	Yes
Data Recovery and High Availability Guide and Reference	SC10-4228	Yes
Developing ADO.NET and OLE DB Applications	SC10-4230	Yes
Developing Embedded SQL Applications	SC10-4232	Yes

Table 21. DB2 technical information (continued)

Name	Form Number	Available in print
Developing SQL and External Routines	SC10-4373	No
Developing Java Applications	SC10-4233	Yes
Developing Perl and PHP Applications	SC10-4234	No
Getting Started with Database Application Development	SC10-4252	Yes
Getting started with DB2 installation and administration on Linux and Windows	GC10-4247	Yes
Message Reference Volume 1	SC10-4238	No
Message Reference Volume 2	SC10-4239	No
Migration Guide	GC10-4237	Yes
Net Search Extender Administration and User's Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6842	Yes
Performance Guide	SC10-4222	Yes
Query Patroller Administration and User's Guide	GC10-4241	Yes
Quick Beginnings for DB2 Clients	GC10-4242	No
Quick Beginnings for DB2 Servers	GC10-4246	Yes
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC18-9749	Yes
SQL Guide	SC10-4248	Yes
SQL Reference, Volume 1	SC10-4249	Yes
SQL Reference, Volume 2	SC10-4250	Yes
System Monitor Guide and Reference	SC10-4251	Yes
Troubleshooting Guide	GC10-4240	No
Visual Explain Tutorial	SC10-4319	No
What's New	SC10-4253	Yes
XML Extender Administration and Programming	SC18-9750	Yes
XML Guide	SC10-4254	Yes
XQuery Reference	SC18-9796	Yes

Table 22. DB2 Connect-specific technical information

Name	Form Number	Available in print
DB2 Connect User's Guide	SC10-4229	Yes

Table 22. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Personal Edition	GC10-4244	Yes
Quick Beginnings for DB2 Connect Servers	GC10-4243	Yes

Table 23. WebSphere Information Integration technical information

Name	Form Number	Available in print
WebSphere Information Integration: Administration Guide for Federated Systems	SC19-1020	Yes
WebSphere Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018	Yes
WebSphere Information Integration: Configuration Guide for Federated Data Sources	SC19-1034	No
WebSphere Information Integration: SQL Replication Guide and Reference	SC19-1030	Yes

Note: The DB2 Release Notes provide additional information specific to your product's release and fix pack level. For more information, see the related links.

Related concepts:

- "Overview of the DB2 technical information" on page 445
- "About the Release Notes" in *Release notes*

Related tasks:

- "Ordering printed DB2 books" on page 448

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* CD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation CD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation CD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Procedure:

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  - Locate the contact information for your local representative from one of the following Web sites:
    - The IBM directory of world wide contacts at www.ibm.com/planetwide
    - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  - When you call, specify that you want to order a DB2 publication.
  - Provide your representative with the titles and form numbers of the books that you want to order.

Related concepts:

- "Overview of the DB2 technical information" on page 445

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 446

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure:

To invoke SQL state help, open the command line processor and enter:

```
? 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, and `? 08` displays help for the 08 class code.

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*

Accessing different versions of the DB2 Information Center

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Related tasks:

- "Updating the DB2 Information Center installed on your computer or intranet server" on page 451

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

Procedure:

To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button.

     Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

To display topics in your preferred language in a Firefox or Mozilla browser:

1. Select the **Tools** —> **Options** —> **Languages** button. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Related concepts:

- "Overview of the DB2 technical information" on page 445

Updating the DB2 Information Center installed on your computer or intranet server

If you have a locally-installed DB2 Information Center, updated topics can be available for download. The 'Last updated' value found at the bottom of most topics indicates the current level for that topic.

To determine if there is an update available for the entire DB2 Information Center, look for the 'Last updated' value on the Information Center home page. Compare the value in your locally installed home page to the date of the most recent downloadable update at http://www.ibm.com/software/data/db2/udb/support/icupdate.html. You can then update your locally-installed Information Center if a more recent downloadable update is available.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to determine if update packages are available from IBM.

   Note: Updates are also available on CD. For details on how to configure your Information Center to install updates from CD, see the related links.

   If update packages are available, use the Update feature to download the packages. (The Update feature is only available in stand-alone mode.)
3. Stop the stand-alone Information Center, and restart the DB2 Information Center service on your computer.

Procedure:

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center service.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv9 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `C:\Program Files\IBM\DB2 Information Center\Version 9` directory.
     c. Run the `help_start.bat` file using the fully qualified path for the DB2 Information Center:

        ```
        <DB2 Information Center dir>\doc\bin\help_start.bat
        ```
   - On Linux:

a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9` directory.
b. Run the `help_start` script using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>/doc/bin/help_start
```

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the Update button (). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click **Install Updates**.
5. After the download and installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center.
   - On Windows, run the `help_end.bat` file using the fully qualified path for the DB2 Information Center:

     ```
     <DB2 Information Center dir>\doc\bin\help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate `help_start.bat`.
   - On Linux, run the `help_end` script using the fully qualified path for the DB2 Information Center:

     ```
     <DB2 Information Center dir>/doc/bin/help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the `help_start` script.
7. Restart the DB2 Information Center service.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv9 start
     ```

The updated DB2 Information Center displays the new and updated topics.

Related concepts:

- "DB2 Information Center installation options" in *Quick Beginnings for DB2 Servers*

Related tasks:

- "Installing the DB2 Information Center using the DB2 Setup wizard (Linux)" in *Quick Beginnings for DB2 Servers*
- "Installing the DB2 Information Center using the DB2 Setup wizard (Windows)" in *Quick Beginnings for DB2 Servers*

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin:

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials:

To view the tutorial, click on the title.

Native XML data store
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

Related concepts:

- "Visual Explain overview" in *Administration Guide: Implementation*

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation
Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Related concepts:

- "Introduction to problem determination" in *Troubleshooting Guide*
- "Overview of the DB2 technical information" on page 445

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix L. 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

Company, product, or service names identified in the documents of the DB2 Version 9 documentation library may be trademarks or service marks of International Business Machines Corporation or other companies. Information on the trademarks of IBM Corporation in the United States, other countries, or both is located at http://www.ibm.com/legal/copytrade.shtml.

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 documentation library:

Microsoft, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Itanium®, Pentium®, and Xeon® 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

Contacting IBM

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts at http://www.ibm.com/planetwide

To learn more about DB2 products, go to http://www.ibm.com/software/data/db2/.

Printed in USA

SC10-4228-00

Spine information:

IBM DB2 **DB2 Version 9** **Data Recovery and High Availability Guide and Reference**