IBM® DB2® Universal Database

Data Recovery and High Availability Guide and Reference

Version 7

DATA-RCVR-00

IBM® DB2® Universal Database

Data Recovery and High Availability Guide and Reference

Version 7

DATA-RCVR-00

Before using this information and the product it supports, be sure to read the general information under "Appendix K. Notices" on page 499.

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

Contents

About This Book

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

Who Should Use this Book

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

It is assumed that you are familiar with DB2 Universal Database, Structured Query Language (SQL), and with the operating system environment in which DB2 UDB is running. For general information about DB2 UDB, see the *Administration Guide*. For information about SQL, see the *SQL Reference*. For information about configuring, invoking, and using the DB2 UDB command line processor, see the *Command Reference*. For information about the DB2 UDB application programming interfaces (APIs), see the *Administrative API Reference*. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*. This manual does not contain instructions for installing DB2, which depend on your operating system. Installation information can be found in the appropriate *Quick Beginnings* book for 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 rebuild damaged or corrupted databases or table spaces that were previously backed up.

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

High Availability

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

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

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

Chapter 8. High Availability in the Solaris Operating Environment
Discusses DB2 support for high availability failover recovery in the Solaris Operating Environment, which is currently implemented through Sun Cluster 2.x (SC2.x), 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 sample programs containing recovery-related DB2 APIs and embedded SQL calls, and information on how to use them.

Appendix F. Recovery CLP Script
Provides the code listing for a DB2 command script containing recovery-related CLP commands, and information on how to use it.

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

Appendix H. 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 I. 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

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

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

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

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

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

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work

(transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

For detailed information about these different recovery methods, see "Version Recovery" on page 22, "Rollforward Recovery" on page 23, or "Crash Recovery" on page 9.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 1). You cannot directly modify a recovery log file or the recovery history file; however, they are important should you need to use your database backup image to recover data that is lost or damaged. Each database includes *recovery logs*, which are used to recover from

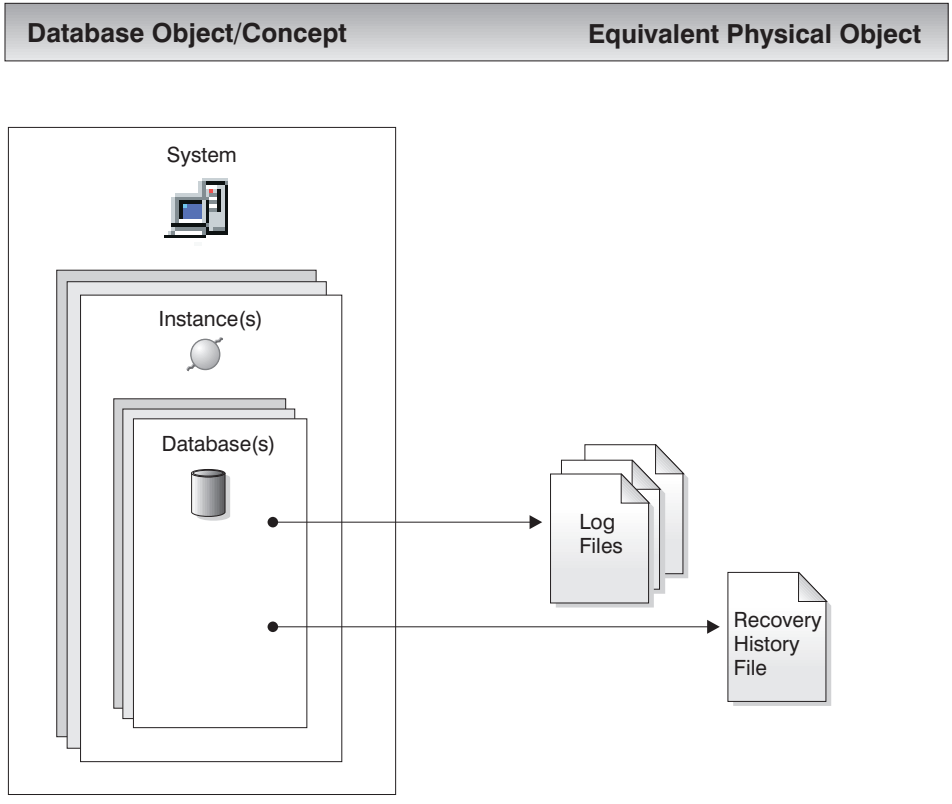


Figure 1. Recovery Log Files and the Recovery History File

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 in case 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.

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

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

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

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or

rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Deciding How Often to Back Up

Your recovery plan should allow for regularly scheduled backup operations, because backing up a database requires time and system resources. Your plan may include a combination of full database backups and incremental backup operations (see "Incremental Backup and Recovery" on page 25).

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

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

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

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

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

Offline backup operations are faster than online backup operations.

If a database contains large amounts of long field and large object (LOB) data, backing up the database could be very time-consuming. 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. By storing long field and LOB data in separate table spaces, the time required to complete a backup operation can be reduced by choosing not to back up the table spaces containing the long field and LOB data. If the long field and LOB data is critical to your business, backing up these table spaces should be considered against the time required to complete the restore operation for these table spaces. 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: Following is a special consideration if you keep your long field data, LOB data, and indexes in separate table spaces, but do not back them up together: If you back up a table space that does not contain all of the table data, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces that contain any type of data for a table must be rolled forward simultaneously to the same point in time.

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

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

Storage Considerations

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

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

Storage Considerations

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

The backup of an SMS table space that contains LOBs can be larger than the size of the original table space. The backup can be as much as 40 per cent larger, depending on the LOB data size in the table space. For example, if you take a backup of a 1-GB SMS table space (with LOBs), you will need more than 1 GB of disk space when you restore it. This only occurs on file systems that support sparse allocation (for example, on UNIX based operating systems).

To prevent media failure from destroying a database and your ability to rebuild it, keep the database backup, the database logs, and the database itself on different devices. For this reason, it is highly recommended that you use the *newlogpath* configuration parameter to put database logs on a separate device once the database is created. (This and other configuration parameters related to logging are discussed in "Configuration Parameters for Database Logging" on page 37.)

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:

- Dedicate enough space in the database log path directory to retain the logs.
- 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. For example, on OS/2, DB2 supports a user exit program to handle the storage of both database backup images and database logs on standard and non-standard devices. (For more information, see "Appendix H. User Exit for Database Recovery" on page 439.)

Keeping Related Data Together

As part of your database design, you will know the relationships that exist between tables. These relationships can be expressed at the application level, when transactions update more than one table, or at the database level, where referential integrity exists between tables, or where triggers on one table affect

another table. You should consider these relationships when developing a recovery plan. You will want to back up related sets of data together. Such sets can be established at either the table space or the database level. By keeping related sets of data together, you can recover to a point where all of the data is consistent. This is especially important if you want to be able to perform point-in-time rollforward recovery on table spaces.

Using Different Operating Systems

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

There is, however, support for cross-platform backup and restore operations between Sun Solaris and HP. When you transfer the backup image between systems, you must transfer it in binary mode. On the target system, the database must be created with the same code page and territory as the system on which the original database was created.

If you must move tables from one operating system to another, and cross-platform backup and restore support is not available in your environment, you can use the **db2move** command, or the export utility followed by the import or the load utility. For more information about these utilities, see the *Data Movement Utilities Guide and 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 10). When a database is in a consistent and usable state, it has attained what is known as a "point of consistency".

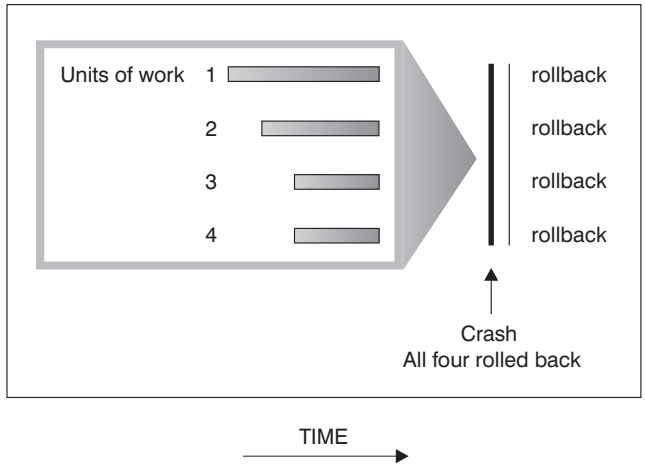


Figure 2. Rolling Back Units of Work (Crash Recovery)

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 and unusable state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A serious operating system error that causes DB2 to go down.

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. For more information about this parameter, see the *Administration Guide: Performance* book.) If you do not want automatic restart behavior, you will need to issue the RESTART DATABASE command when a database failure occurs. The `db2diag.log` file records when the database restart operation begins.

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logretain* configuration parameter is set to `RECOVERY`, or the *userexit* configuration parameter is enabled), and an error occurs during crash recovery that is attributable to an individual table space, that table space must 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 are generally still usable, and connections to the database can be established.

Recovering Damaged Table Spaces

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

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

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

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

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

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

Recovering Table Spaces in Recoverable Databases

The damaged table space is put in offline and not accessible state, and in rollforward pending state, because crash recovery is necessary. The restart operation will succeed if there is no additional problem. The damaged table space can be used again once you:

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

Recovering Table Spaces in Non-recoverable Databases

Since crash recovery is necessary, and logs are not kept indefinitely, the restart operation can only succeed if the user is willing to drop the damaged table spaces. (Successful completion of recovery means that the log records

necessary to recover the damaged table spaces to a consistent state will be gone; therefore, the only valid action against such table spaces is to drop them.)

You can do this by invoking an unqualified restart database operation. It will succeed if there are no damaged table spaces. If it fails (SQL0290N), you can look in the `db2diag.log` file for a complete list of table spaces that are currently damaged.

- If you are willing to drop all of these table spaces once the restart database operation is complete, you can initiate another restart database operation, listing all of the damaged table spaces under the DROP PENDING TABLESPACES option. If a damaged table space is included in the DROP PENDING TABLESPACES list, the table space is put into drop pending state, and your only option after recovery is to drop the table space. The restart operation continues without recovering this table space. If a damaged table space is *not* included in the DROP PENDING TABLESPACES list, the restart database operation fails with SQL0290N.
- If you are unwilling to drop (and thus lose the data in) these table spaces, your options are to:
  - Wait and fix the damaged containers (without losing the original data), and then try the restart database operation again
  - Perform a database restore operation.

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

Reducing the Impact of Media Failure

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

- Mirror or duplicate the disks that hold the data and logs for important databases.
- Use a Redundant Array of Independent Disks (RAID) configuration, such as RAID Level 5. For more information about RAID, see “Protecting Against Disk Failure” on page 13.
- In a partitioned database environment, set up a rigorous procedure for handling the data and the logs on the catalog node. Because this node is critical for maintaining the database:
  - Ensure that it resides on a reliable disk

- Duplicate it
- Make frequent backups
- Do not put user data on it.

Protecting Against Disk Failure

If you are concerned about the possibility of damaged data or logs due to a disk crash, consider the use of some form of disk fault tolerance. Generally, this is accomplished through the use of a *disk array*. A disk array consists of a collection of disk drives that appear as a single large disk drive to an application.

Disk arrays involve *disk striping*, which is the distribution of a file across multiple disks, the mirroring of disks, and data parity checks.

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

For detailed information about the available RAID levels, visit:

```
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 (see "Hardware Disk Arrays" on page 13), 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 (see "Hardware Disk Arrays" on page 13). Although redundant disks are required, a software disk array is comparatively inexpensive to implement, because costly disk controllers are not required.

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

Reducing the Impact of Transaction Failure

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

- An uninterrupted power supply

- Adequate disk space for database logs
- Reliable communication links among the database partition servers in a partitioned database environment
- Synchronization of the system clocks in a partitioned database environment (see "Synchronizing Clocks in a Partitioned Database System" on page 140).

Recovering from Transaction Failures in a Partitioned Database Environment

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

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

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

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

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

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

For more information about two-phase commit, see the *Administration Guide: Planning* book.

Transaction Failure Recovery on an Active Database Partition Server

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

- If the still active database partition server is the coordinator node for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.
- If the failed database partition server was the coordinator node for the application, agents that are still working for the application on the active servers are interrupted to do failure recovery. The current transaction is rolled back locally on each server, unless it has been prepared and is waiting for the transaction outcome. In this situation, the transaction is left in doubt on the active database partition servers, and the coordinator node is not aware of this (because it is not available).

  For more information about how an indoubt transaction is resolved, see the *Administration Guide: Planning* book.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator node, agents working for this application are interrupted. The coordinator node will either send a ROLLBACK or a disconnect message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator node returns SQL0279.

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

Transaction Failure Recovery on the Failed Database Partition Server

If the transaction failure causes the database manager to end abnormally, you can issue the **db2start** command with the RESTART option to restart the database manager once the processor has restarted. If you cannot restart the processor, you can issue **db2start** to restart the database manager on a different processor. For more information, see the *Command Reference*.

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

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

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

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

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

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

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

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

Note: The LIST INDOUBT TRANSACTIONS command is also used in a distributed transaction environment. To distinguish between the two

types of indoubt transactions, the *originator* field in the output that is returned by the LIST INDOUBT TRANSACTIONS command displays one of the following:

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

For more information about distributed environments, see the *Administration Guide: Planning* book.

Identifying the Failed Database Partition Server

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

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

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

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

Determining which database partition server failed is a two-step process. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the `db2nodes.cfg` file.) On the database partition server that detects the error, a message that indicates the node number of the failed server is written to the `db2diag.log` file.

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

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.

Recovering Indoubt Transactions on the Host

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

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

Recovery when DB2 Connect Has the DB2 Syncpoint Manager Configured

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

1. Connect to the SPM as shown below:

```
db2 => connect to db2spm

 Database Connection Information

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

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

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

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

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

   then the transaction was rolled back.

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

Recovery when DB2 Connect Does Not Use the DB2 Syncpoint Manager

Use the information in this section when TCP/IP connectivity is used to update DB2 for OS/390 in a multisite update from either DB2 Connect Personal Edition or DB2 Connect Enterprise 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. For more information about alerts, see the *Troubleshooting Guide*.

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

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

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

From this list identify the transaction that you want to heuristically complete. For details about the DISPLAY command, see the *DB2 for OS/390 Command Reference*. The LUWID displayed can be matched to the same luwid at the Transaction Manager Database.

2. Issue the RECOVER THREAD(<LUWID>) ACTION(ABORT|COMMIT) command, depending on what you want to do.

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

Disaster Recovery

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

- A site to be used in the event of an emergency
- A different machine on which to recover the database
- Off-site storage of database backups and archived logs.

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

The usefulness of a table space backup for disaster recovery depends on the scope of the failure. Typically, disaster recovery requires that you restore the entire database; therefore, a full database backup should be kept at a standby site. Even if you have a separate backup image of every table space, you cannot use them to recover the database. If the disaster is a damaged disk, a table space backup of each table space on that disk can be used to recover. If you have lost access to a container because of a disk failure (or for any other reason), you can restore the container to a different location. For additional information, see “Redefining Table Space Containers During a Restore Operation (Redirected Restore)” on page 107.

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

Version Recovery

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

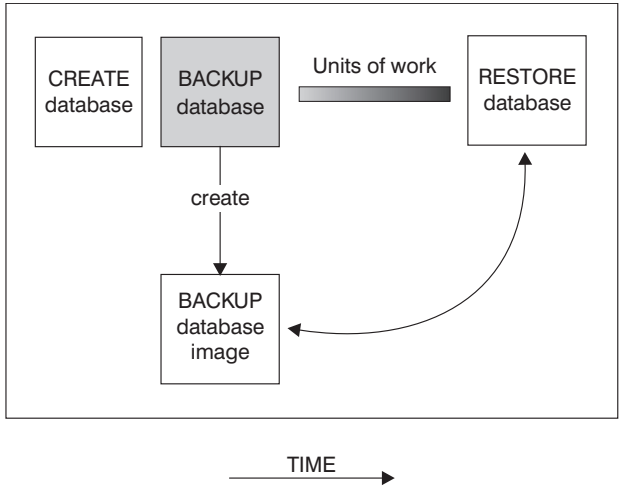


Figure 3. Version Recovery. The database is restored from the latest backup image, but all units of work processed between the time of backup and failure are lost.

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

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

Rollforward Recovery

To use the *rollforward recovery* method, you must have taken a backup of the database, and archived the logs (by enabling either the *logretain* or the *userexit* database configuration parameters, or both. For information on the decisions that you must make regarding the logging procedure that you use, see "Understanding Recovery Logs" on page 30.) 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.

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 24). The database logs record all changes made to the database. This method completes the recovery of the database to its state at a particular point in time, or to its state immediately before the failure (that is, to the end of the active logs.)

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

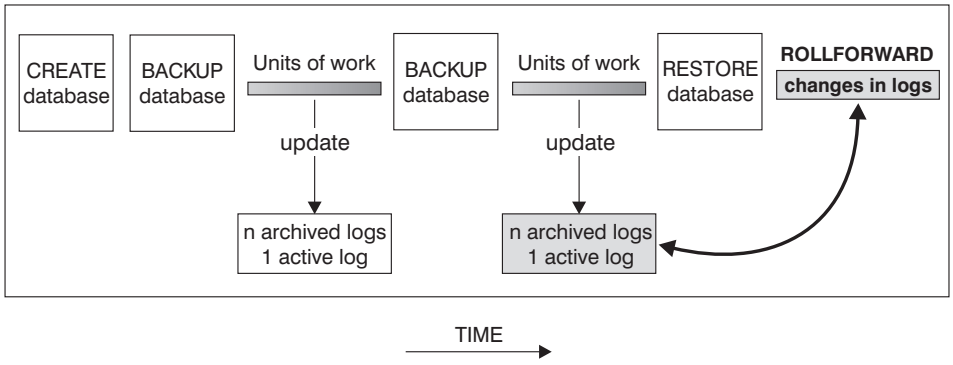


Figure 4. Database Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

- *Table space rollforward recovery*. If the database is enabled for forward recovery, it is also possible to back up, restore, and roll table spaces forward (see Figure 5 on page 25). 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 (see "ROLLFORWARD DATABASE Command" on page 142) to apply the logs against the table spaces to either a point in time, or to the end of the logs.
- If one or more table spaces are in *rollforward pending* state after crash recovery, first correct the table space problem. In some cases, correcting the table space problem does not involve a restore database operation. For example, a power loss could leave the table space in rollforward pending state. A restore database operation is not required in this case. Once the problem with the table space is corrected, you can use the ROLLFORWARD DATABASE command to apply the logs against the table spaces to the end of the logs. If the problem is corrected before crash recovery, crash recovery may be sufficient to take the database to a consistent, usable state.

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

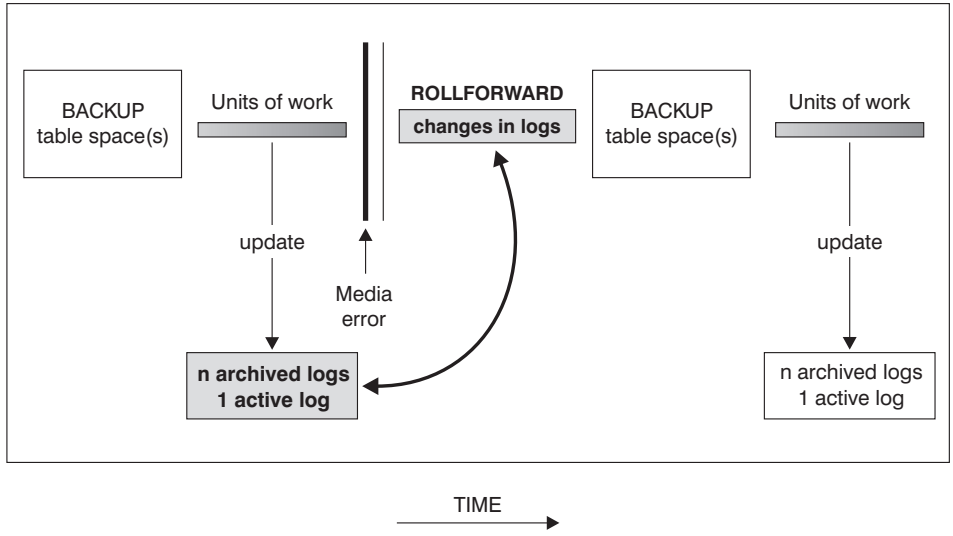


Figure 5. Table Space Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

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

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

Incremental Backup and Recovery

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

- When a small percentage of the data in a warehouse changes, it should not be necessary to back up the entire database.

- Appending table spaces to existing databases and then taking only table space backups is risky, because there is no guarantee that nothing outside of the backed up table spaces has changed between table space backups.

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

Two types of incremental backup are supported:

- *Incremental*. An incremental backup image is a copy of all database data that has changed since the most recent, successful, full backup operation. This is also known as a cumulative backup image, because a series of incremental backups taken over time will each have the contents of the previous incremental backup image. The predecessor of an incremental backup image is always the most recent successful full backup of the same object.
- *Delta*. A delta, or incremental delta, backup image is a copy of all database data that has changed since the last successful backup (full, incremental, or delta) of the table space in question. This is also known as a differential, or non-cumulative, backup image. The predecessor of a delta backup image is the most recent successful backup containing a copy of each of the table spaces in the delta backup image.

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

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

To rebuild the database or the table space to a consistent state, the recovery process must begin with a consistent image of the entire object (database or table space) to be restored, and must then apply each of the appropriate

incremental backup images in the order described below (see "Restoring from Incremental Backup Images").

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.
- YES. Incremental backup is permitted with this configuration. When update tracking is enabled, the change becomes effective at the first successful connection to any database in the instance. A full database backup is necessary before an incremental backup can be taken.

The default *trackmod* setting for existing databases is `NO`; for new databases, it is `YES`.

For SMS table spaces, the granularity of this tracking is at the table space level. For DMS table spaces, the granularity is at the extent level for data and index pages, and at the table space level for other page types.

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

Restoring from Incremental Backup Images

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

1. Identifying the incremental target image.

   The DBA must first 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. An incremental restore command against this image may initiate the creation of a new database with the configuration and table space definitions from this target image. 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.

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

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

This will result in the DB2 restore utility performing each of the steps described above 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 restore will not be possible, and you will have to proceed with a manual restore procedure.

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

If the database history is not available, you can perform an incremental restore operation manually, following the steps outlined at the beginning of this section. For example:

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

   where:
     <ts> points to the last incremental backup 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 <
```

In cases where a database restore operation is being attempted, and table space incremental backup images have been produced, the table space images must be restored in the chronological order of their backup time stamps.

Limitations to Automatic Incremental Restore

1. If you renamed a table space since the backup you wish to restore from and you issue a table space level restore using the new name, the required chain of backup images using the database history will not be generated correctly and an error will occur.

   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>
   ```

   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.

   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.

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 three types of DB2 logging: *circular*, *capture*, and *archive*, each providing a different level of recovery capability:

- *Circular* logging is the default behavior when a new database is created. (The *logretain* database configuration parameter setting is `NO`.) With this type of logging, only full, offline backups of the database are valid. 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. The database must be offline (inaccessible to users) when a full backup is taken. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version recovery*.

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

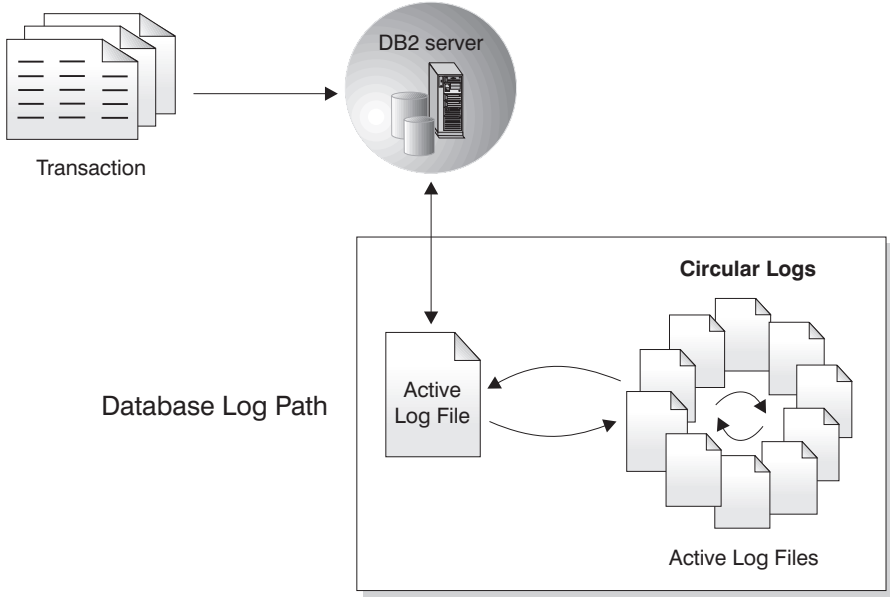


Figure 6. Circular Logging

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

- *Capture* logging can be configured by setting the *logretain* database configuration parameter to `CAPTURE`. Capture logging is used for replication processing. Log files are retained until replication processing has completed, at which time they are deleted automatically. All DB2 utilities handle this logging mode in the same way as they handle circular logging; that is, neither online backup operations, nor table space backup and restore operations, nor rollforward operations are allowed; and a load operation specifying the RECOVERY NO option will not put table spaces in backup pending state.
- *Archive* logging is used specifically for rollforward recovery. It can be configured by setting the *logretain* database configuration parameter to `RECOVERY`. Archived logs can be:

  online archived logs
  When changes in the active log are no longer needed for normal processing, the log is closed, and becomes an archived log. An archived log is said to be *online* when it is stored in the database log path directory (see Figure 7 on page 32).

  offline archived logs
  An archived log is said to be *offline* when it is no longer found in the database log path directory (see Figure 8 on page 33). You can also store archived logs in a location other than the database log path directory by using a user exit program. (For additional information, see "Appendix H. User Exit for Database Recovery" on page 439.)

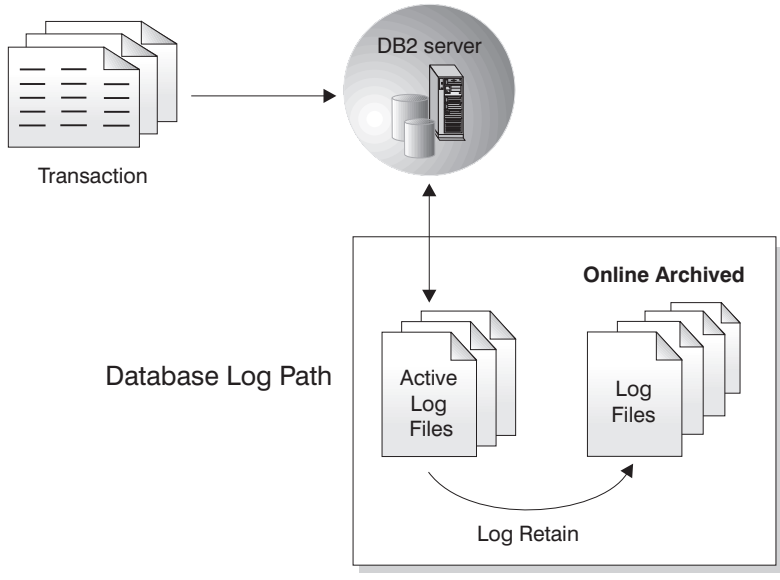


Figure 7. Online Archive Logging

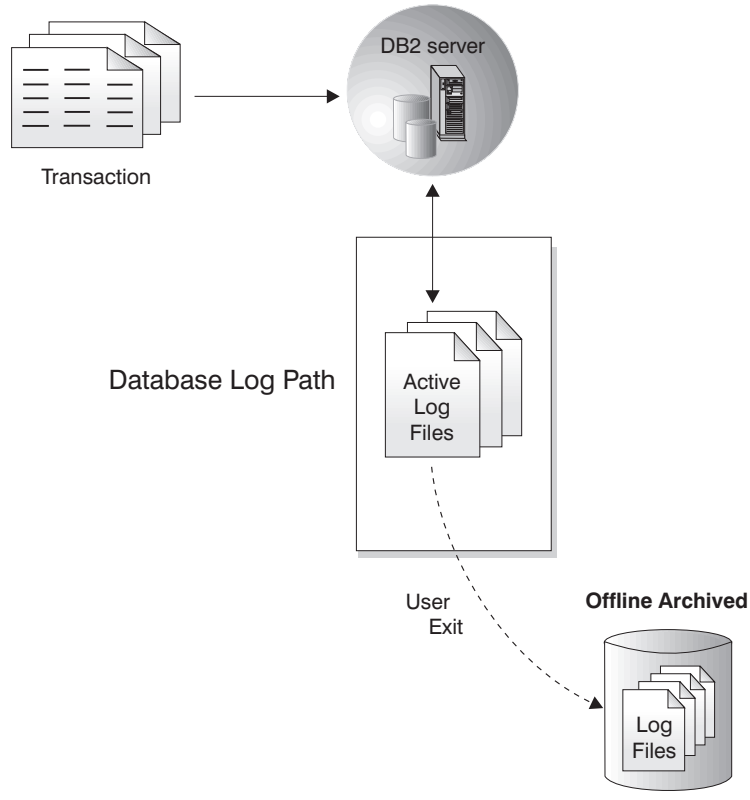
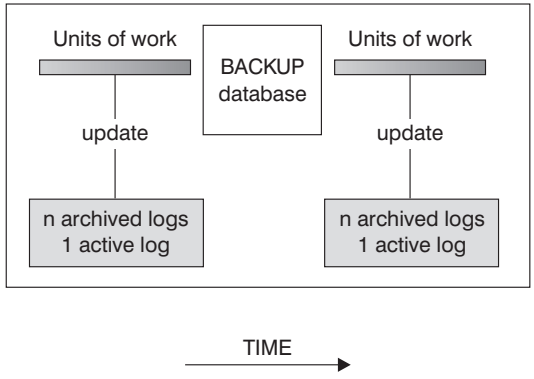


Figure 8. Offline Archive Logging

Rollforward recovery can use both archived logs and active logs to rebuild a database either to the end of the logs, or to a specific point in time. The rollforward utility achieves this by reapplying committed changes found in the archived and active logs to the restored database.

Rollforward recovery can also use logs to rebuild a table space by re-applying committed updates in both archived and active logs. You can recover a table space to the end of the logs, or to a specific point in time.

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

Two database configuration parameters allow you to change where archived logs are stored: The *newlogpath* parameter, and the *userexit* parameter. Changing the *newlogpath* parameter also affects where active logs are stored. For more information about these configuration parameters, see the *Administration Guide: Performance* book.

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

Notes:

1. If you erase an active log, the database becomes unusable and must be restored before it can be used again. You will be able to roll forward only up to the first log that was erased.
2. If you are concerned that your active logs may be damaged (as a result of a disk crash), you should consider mirroring, either at the operating system level by mirroring the disk used for logging, or at the DB2 level by using the NEWLOGPATH2 registry variable.

Log Mirroring

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

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

If you are concerned that your active logs may be damaged (as a result of a disk crash), you should consider using a new DB2 registry variable, NEWLOGPATH2, 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 NEWLOGPATH2 registry variable 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.

Note: Because Windows NT does not allow "mounting" a device under an arbitrary path name, it is not possible (on this platform) to specify a secondary path on a separate device.

NEWLOGPATH2 can be enabled (set to 1) or disabled (set to 0). The default value is zero. If this variable is set to 1, the secondary path name is the current value of the LOGPATH variable concatenated with the character 2. For example, in an SMP environment, if LOGPATH is `/u/dbuser/sqllogdir/logpath`, the secondary log path will be `/u/dbuser/sqllogdir/logpath2`. In an MPP environment, if LOGPATH is `/u/dbuser/sqllogdir/logpath`, DB2 will append the node indicator to the path and use `/u/dbuser/sqllogdir/logpath/NODE0000` as the primary log path. In this case, the secondary log path will be `/u/dbuser/sqllogdir/logpath2/NODE0000`.

When NEWLOGPATH2 is first enabled, it will not actually be used until the current log file is completed on the next database startup. This is similar to how LOGPATH and NEWLOGPATH are currently used.

If there is an error writing to either path 1 or path 2, the database will mark the failing path as "bad", write a message to the `db2diag.log` file, and write subsequent log records to the remaining "good" log path only. DB2 will not attempt to use the "bad" path again until the current log file is completed. When DB2 needs to open the next log file, it will verify that this path is valid, and if so, will begin to use it. If not, DB2 will not attempt to use the path again until the next log file is accessed for the first time. There is no attempt to synchronize the log paths, but DB2 keeps information about access errors that occur, so that the correct paths are used when log files are archived. If a failure occurs while writing to the remaining "good" path, the database abends.

Reducing Logging on Work Tables

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

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 parameter:

- *All* changes to the table must be flushed out to disk at commit time. This means that the commit may take longer.
- An error returned for any operation in a unit of work in which the table is created will result in the rollback of the entire unit of work (SQLCODE -1476, SQLSTATE 40506).
- You cannot recover these tables when rolling forward. If the rollforward operation encounters a table that was created 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. For more information, see the "Concurrency" section of the *Administration Guide: Performance* book.

For more information about creating tables, see the *SQL Reference*.

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).

For more information about declared temporary tables and their limitations, see the DECLARE GLOBAL TEMPORARY TABLE statement in the *SQL Reference*.

Configuration Parameters for Database Logging

The database configuration file contains parameters related to rollforward recovery. The default values of these parameters do not support this type of recovery, so if you plan to use it, you must change some of the default values. For more information about configuring DB2 Universal Database, see the *Administration Guide: Performance* book.

Primary logs (logprimary)

This parameter specifies the number of primary logs 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.

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.

The total log file size limit is 32 GB. That is, the number of log files (*logprimary* + *logsecond*) multiplied by the size of each log file, in bytes (*logfilsiz* * 4096) must be less than 32 GB.

For more information about this configuration parameter, see the *Administration Guide: Performance* book.

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. An error is returned, and activity against the database is stopped, if more than the configured number of secondary log files is required.

For more information about this configuration parameter, see the *Administration Guide: Performance* book.

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

There is a 32-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 65535, and the upper limit on (*logprimary* + *logsecond*), which is 128. Thus, ((*logprimary* + *logsecond*) * *logfilsiz*) < (32 GB / 4096).

The size of the log file has a direct bearing on performance. If the database is configured to retain logs, each time a log is filled, a request is issued for allocation and initialization of a new log. Increasing the size of the log reduces the number of requests required to allocate and initialize new logs. (Keep in mind, however, that with a larger log size it takes more time to format each new log). The formatting of new logs is transparent to applications connected to the database, so that database performance is unaffected by formatting.

Assuming that you have an application that keeps the database open to minimize processing time when opening the database (see "Enhancing Recovery Performance" on page 54), the log file size should be determined by the amount of time it takes to make offline archived log copies.

The data transfer speed of the device you use to store offline archived logs, and the software used to make the copies, must at a minimum match the average rate at which the database manager writes data in the logs. If the transfer speed cannot keep up with new log data being generated, you may run out of disk space if logging activity continues for a sufficiently long period of time, 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. (For information about using different media for

storing logs, see "Appendix H. User Exit for Database Recovery" on page 439 .) Some tape devices require the same amount of time to copy a file, regardless of its size. You must determine the capabilities of your archiving device.

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

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

Log Buffer (logbufsz)

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

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

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

Number of Commits to Group (mincommit)

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

The grouping of commits occurs only if the value of this parameter is greater than 1, and if the number of applications connected to the database is greater than the value of this parameter. When commit

grouping is in effect, application commit requests are held until either one second has elapsed, or the number of commit requests equals the value of this parameter.

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

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

You must keep track of the location of the logs.

Changes are not applied until the database is in a consistent state. The configuration parameter *database_consistent* returns the status of the database. For more information about this configuration parameter, see the *Administration Guide: Performance* book. For information about the roles that database logs play if a database is not in a consistent state, see "Managing Log Files".

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

If logretain is set to `CAPTURE`, the replication capture program calls the PRUNE LOGFILE command to delete log files when the capture program completes. You should not set logretain to `CAPTURE` if you want to perform rollforward recovery on the database.

User exit (userexit)
This parameter causes the database manager to call a user exit program for archiving and retrieving logs. The log files are archived in a location that is different from the active log path. If *userexit* is set to `ON`, rollforward recovery is enabled. For information about user exit programs, see "Appendix H. User Exit for Database Recovery" on page 439.

Managing Log Files

Consider the following when managing database logs:

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

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

  When a rollforward operation completes successfully, the last log that was used is truncated, and logging begins with the next sequential log. Any log in the log path directory with a sequence number greater than the last log used for rollforward recovery is re-used. Any entries in the truncated log following the truncation point are overwritten with zeros. Ensure that you make a copy of the logs before invoking the rollforward utility. (You can invoke a user exit program to copy the logs to another location. For information about user exit programs, see "Appendix H. User Exit for Database Recovery" on page 439.)
- If a database has not been activated (by way of the ACTIVATE DATABASE command), DB2 truncates the current log file when all applications have disconnected from the database. The next time an application connects to the database, DB2 starts logging to a new log file. If many small log files are being produced on your system, you may want to consider using the ACTIVATE DATABASE command. This not only saves the overhead of having to initialize the database when applications connect, it also saves the overhead of having to allocate a large log file, truncate it, and then allocate a new large log file.
- An archived log may be associated with two or more different *log sequences* for a database, because log file names are reused (see Figure 10 on page 42). 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 in the old log sequence are invalid. The restore utility fails to recognize a table space backup image on an old log sequence if a database restore operation is immediately followed by the table space restore operation. Until the database is actually rolled forward, the log

sequence that is to be used is unknown. If the table space is on an old log sequence, it must be "caught" by the table space rollforward operation. A restore operation using an invalid backup image may complete successfully, but a table space rollforward operation will fail, and the table space will be left in rollforward 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 10). 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.

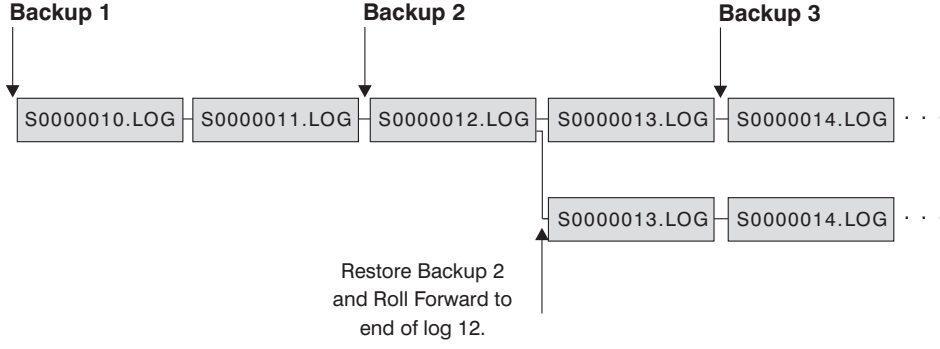


Figure 10. Re-using Log File Names

Managing Log Files with a User Exit Program

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

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

  Note: On Windows NT, you cannot use a REXX user exit to archive logs.

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

  However, if you are moving the data to a slower device such as tape, you might want to have larger log files to prevent the queue from building up. If the queue becomes full, archive and retrieve requests will not be processed. Processing will resume when there is room on the queue. Unprocessed requests will not be automatically requeued.
- An archive request to the user exit program occurs only if *userexit* is configured, and each time an active log file is filled. It is possible that an active log file is not full when the last disconnection from the database occurs and the user exit program is also called for a partially filled active log file.

  Note: To free unused log space, the log file is truncated before it is archived.

- A copy of the log should be made to another physical device so that the offline log file can be used by rollforward recovery if the device containing the log file experiences a media failure. This should not be the same device containing database data files.
- In some cases, if a database is closed before a positive response has been received from a user exit program for an archive request, the database manager will send another request when the database is opened. Thus, a log file may be archived more than once.
- If a user exit program receives a request to archive a file that does not exist (because there were multiple requests to archive and the file was deleted after the first successful archiving operation), or to retrieve a file that does not exist (because it is located in another directory or the end of the logs has been reached), it should ignore this request and pass a successful return code.
- The user exit program should allow for the existence of different log files with the same name after a point in time recovery; it should be written to preserve both log files and to associate those log files with the correct recovery path (see "Managing Log Files" on page 40).
- If two or more databases are using a device at the same time, and one of the operations involves a rollforward operation, a log file needed for rollforward recovery may not exist on the medium currently in the drive. Two conditions can occur:
  - If the user exit program passes a zero (successful) return code back to the database manager, and the requested log file has not been retrieved, the database manager assumes the rollforward operation is complete to the end of the logs, and the rollforward operation stops. However, rollforward processing may not have gone to the end of the logs.
  - If a non-zero return code is returned, the database will be in rollforward pending state, and you must either resume or stop rollforward processing.

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

Blocking Transactions When the Log Directory is Full

A new DB2 registry variable, DB2_BLOCK_ON_LOG_DISK_FULL, can be set to prevent "disk full" errors from being generated when DB2 cannot create a new log file in the active log path.

Instead, DB2 attempts to create the log file every five minutes until it succeeds. If the database is configured with the *userexit* parameter set to `ON`, DB2 also checks for the completion of log file archiving. If an inactive 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

db2diag.log file. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the db2diag.log file.

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

On Demand Log Archive

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

You can initiate on demand log archiving by invoking the ARCHIVE LOG command, or by calling the **db2ArchiveLog** API; these are described in "ARCHIVE LOG" on page 315 and "db2ArchiveLog - Archive Active Log API" on page 328, respectively.

Using Raw Logs

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

- The advantages are:
  - You can attach more than 26 physical drives to a system.
  - The file I/O path length is shorter. This may improve performance on your system. You should conduct benchmarks to evaluate if there are measurable benefits for your work load.
- The disadvantages are:
  - The device cannot be shared by other applications; the entire device *must* be assigned to DB2.
  - The device cannot be operated upon by any operating system utility or third-party tool which would backup or copy from the device.
  - You can easily wipe out the file system on an existing drive if you specify the wrong physical drive number.

You can configure a raw log with the *newlogpath* database configuration parameter. For an example of the syntax used to specify a raw device, see the "Raw I/O" section of the *Administration Guide: Implementation* book. Before doing so, however, consider the advantages and disadvantages listed above, and the additional considerations listed below:

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

If you use multiple disks, this will provide a larger device, and the striping that results can improve performance by faster I/O throughput.

- DB2 will attempt to write to the last 4-KB page of the device. If the device size is greater than 2 GB, the attempt to write to the last page will fail on operating systems that do not provide support for devices larger than 2 GB. In this situation, DB2 will attempt to use all pages, up to the supported limit.

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

  The first 4-KB page of the device is not used by DB2 (this space is generally used by operating system for other purposes.) This means that the total space available to DB2 is *device-size = device-size-available - 1*.
- The *logsecond* parameter is not used. DB2 will not allocate secondary logs. The size of active log space is the number of 4-KB pages that result from *logprimary* x *logfilsiz*.
- Log records are still grouped into log extents, each with a log file size (*logfilsiz*) of 4-KB pages. Log extents are placed in the raw device, one after another. Each extent also consists of an extra two pages for the extent header. This means that the *number of available log extents* the device can support is *device-size* / (*logfilsiz* + 2)
- The device must be large enough to support the active log space. That is, the *number of available log extents* must be greater than (or equal to) the value specified for the *logprimary* configuration parameter.
- If you are using circular logging, the *logprimary* configuration parameter will determine the number of log extents that are written to the device. This may result in unused space on the device.
- If you are using log retention (*logretain*) without a user exit program, after the *number of available log extents* are all used up, all operations that result in an update will receive a log full error. At this time, you must shut down the database and take an offline backup of it to ensure recoverability. After the database backup operation, the log records written to the device are lost. This means that you cannot use an earlier database backup image to restore the database, then roll it forward. If you take a database backup before the *number of available log extents* are all used up, you can restore and roll the database forward.
- If you are using log retention (*logretain*) with a user exit program, the user exit program is called for each log extent as it is filled with log records. The user exit program must be able to read the device, and to store the archived log as a file. DB2 will not call a user exit program to retrieve log files to a raw device. Instead, during rollforward recovery, DB2 will read the extent headers to determine if the raw device contains the required log file. If the required log file is not found in the raw device, DB2 will search the

overflow log path. If the log file is still not found, DB2 will call the user exit program to retrieve the log file into the overflow log path. If you do not specify an overflow log path for the rollforward operation, DB2 will not call the user exit program to retrieve the log file. For additional information about calling a user exit program, see "Calling Format" on page 441.

- If you are using DPROP and writing logs to a raw device, the read log API will not call the user exit program to retrieve log files. Requested log records, however, will be still be returned if they are available on the device. If you request logs that pre-date the oldest ones on the device, they will not be returned (the behavior is similar to DB2 not being able to find the log file that contains the requested log records).

  Notes:

  1. It is recommended that you do not use DPROP when you use a raw device for logging.
  2. If you use the **sqlurlog** API, you should not use a raw device for logging.

Losing Logs

Dropping a database erases all logs in the current database log path directory. Before dropping a database, consider making copies of the logs.

If you are rolling a database forward to a specific point in time, the last log used and all existing logs following that are reused: You lose the ability to recover beyond that particular point in time. Therefore, you should copy all of the logs in the current database log path directory *before* beginning a point-in-time recovery operation.

When the rollforward operation completes, the log file with the last committed transaction is truncated, and logging begins with the next sequential log. If you do not have a copy of the log before it is truncated, and copies of logs with higher sequence numbers, you cannot recover the database beyond the specified point in time. (Once normal database activity resumes, new logs are created, which can be used in any subsequent recovery operation.)

If you change the log path directory and then remove the subdirectory or erase any logs in that subdirectory called for in the log path, the database manager will look for the logs in the default log path, SQLOGDIR, when the database is opened. If the logs are not found, the database is put in backup pending state, and you must back up the database before it is usable. This backup must be made even if the subdirectory contained empty logs.

If you lose the log containing the point in time of the end of the online backup and you are rolling the corresponding restored image forward, the

database will not be usable. To make the database usable, you must restore the database from a different backup and all associated logs.

You may encounter a situation similar to the following: You would like to do a point in time recovery on a full database but you are concerned that you might lose a log during the recovery process. (This scenario could occur if you have an extended number of archived logs between the time of the last backup database image and the point in time where you would like to have the database recovered.)

First, you should copy all of the applicable logs to a "safe" location. Then you can run the RESTORE command and use the rollforward recovery method to the point in time you wish for the database. If any of the logs that you need is damaged or lost during this process, you have a backup copy of all of the logs elsewhere.

Understanding the Recovery History File

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

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

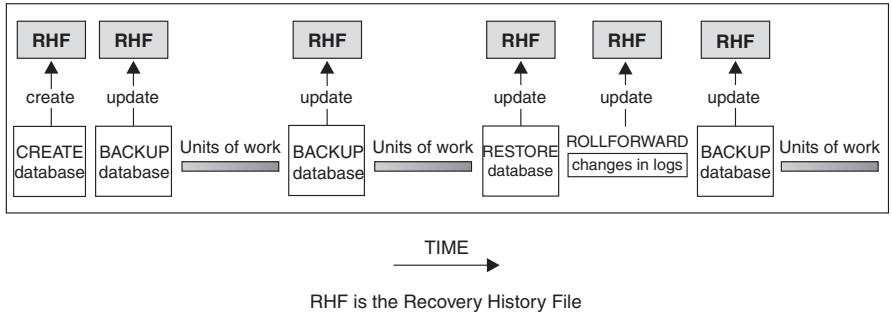


Figure 11. Creating and Updating the Recovery History File

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. For more information about this command, see "LIST HISTORY" on page 318.

Note: When a restore and then a rollforward operation is carried out to the end of logs, the backup ID shown following invocation of the LIST HISTORY command represents the end of time; that is, the backup ID value is 99991231235959. The backup ID is only transformed in this way when a rollforward operation is carried out.

Every backup operation (database, table space, or incremental) includes a copy of the recovery history file. The recovery history file is linked to the database. Dropping a database deletes the recovery history file. Restoring a database to a new location restores the recovery history file. Restoring does not overwrite the existing history recovery file.

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. For more information about this configuration parameter, see the *Administration Guide: Performance* book.

Garbage Collection

Although you can use the PRUNE HISTORY command (see "PRUNE HISTORY/LOGFILE" on page 321) 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 database backup operation completes successfully; it is also invoked after a database restore operation completes successfully. The configuration parameter *num_db_backups* defines how many active full (not 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 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.

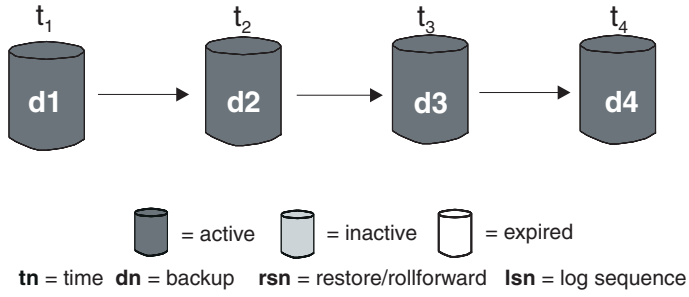


Figure 12. Active Database Backups. The value of *num_db_backups* has been set to four.

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".
If an active database backup image is restored, but it is not the most recent

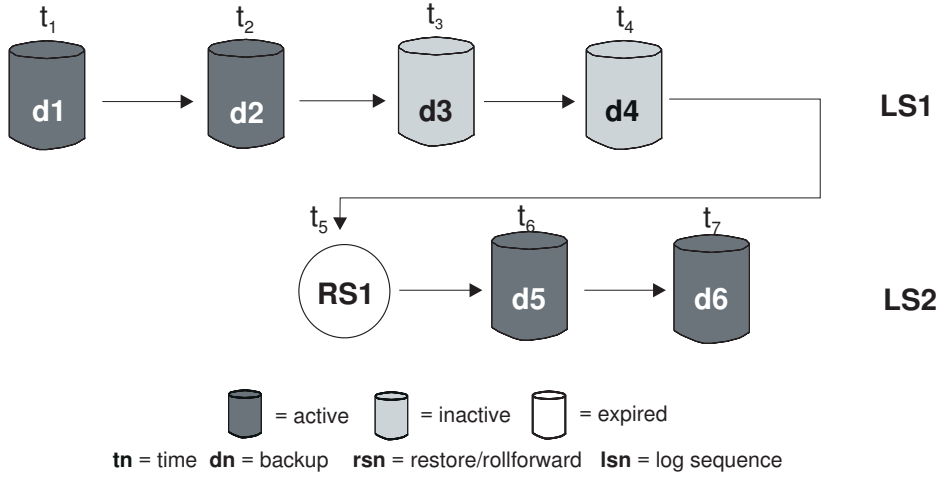


Figure 13. Inactive Database Backups

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”.

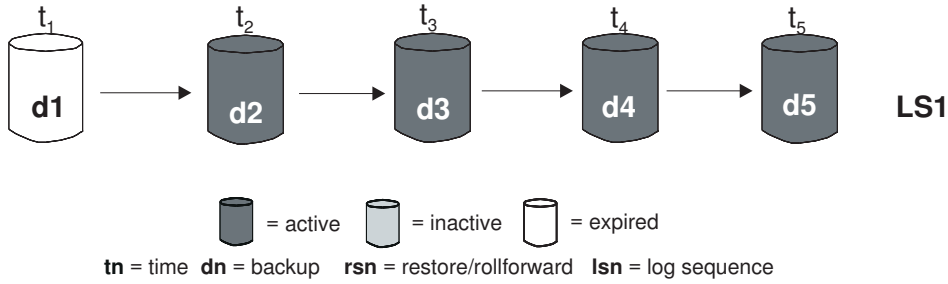


Figure 14. Expired Database Backups

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

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

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

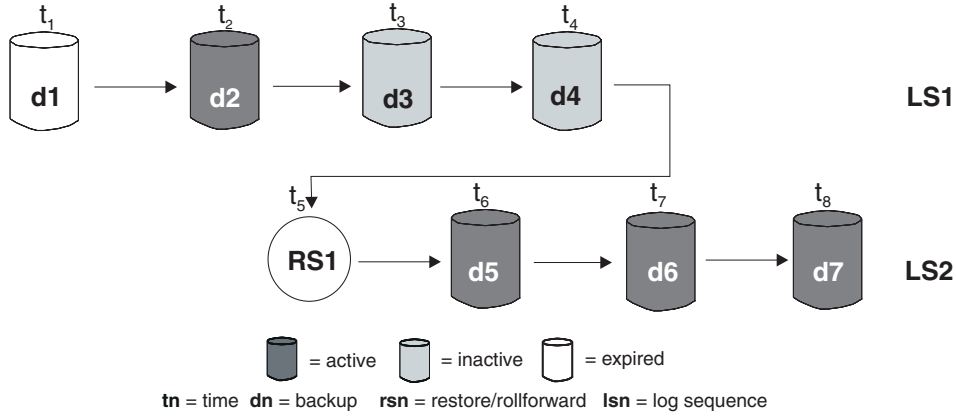


Figure 15. Mixed Active, Inactive, and Expired Database Backups

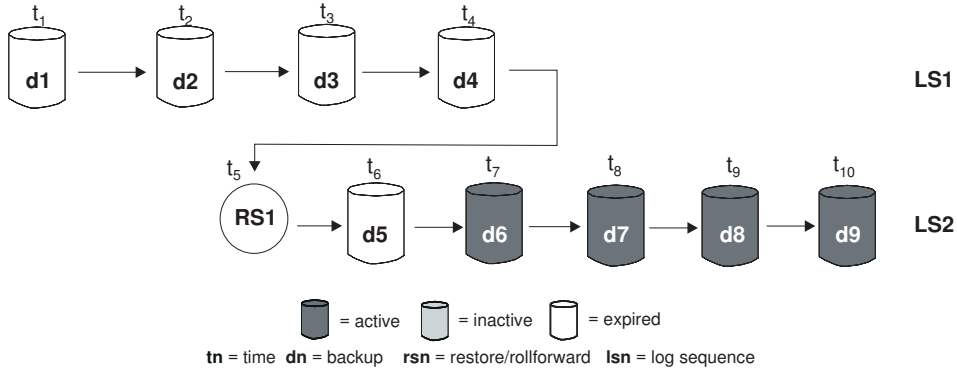


Figure 16. Expired Log Sequence

Understanding Table Space States

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

- *Rollforward pending*. A table space is put in this state after it is restored, or following an input/output (I/O) error. After it is restored, the table space can be rolled forward to the end of the logs or to a point in time. Following an I/O error, the table space must be rolled forward to the end of the logs.
- *Rollforward-in-progress*. A table space is put in this state when a rollforward operation on that table space is in progress. Once the rollforward operation completes successfully, the table space is no longer in rollforward-in-progress state. The table space can also be taken out of this state if the rollforward operation is cancelled.

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

Enhancing Recovery Performance

The following should be considered when thinking about recovery performance:

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

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

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

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

    The *minimum* recommended number of buffers is the number of media devices or containers plus the number specified for the PARALLELISM option.
  - Use multiple source devices.
  - Set the PARALLELISM option for the restore operation to be at least one (1) greater than the number of source devices.
- If a table contains large amounts of long field and LOB data, restoring it could be very time consuming. If the database is enabled for rollforward recovery, the RESTORE command provides the capability to restore selected

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

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

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

Parallel Recovery

DB2 now 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 new agent type introduced by this enhancement is db2agnsc. DB2 chooses the number of agents to be used for database recovery based on the number of CPUs on the machine. For SMP machines, the number of agents used is (number of CPUs + 1). On a machine with a single CPU, multiple agents are used for more efficient reading of logs, processing of log records, and prefetching of data pages.

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.

DB2 Data Links Manager Considerations

The following sections provide information that applies if you have tables that contain DATALINK columns. For a full description of DATALINK columns, see the CREATE TABLE statement in the *SQL Reference*.

Crash Recovery Considerations

When an application issues SQL requests involving Data Links servers running the DB2 Data Links Manager (using DATALINK columns with the FILE LINK CONTROL attribute), the database manager distributes the work to the Data Links servers. It also keeps track of which Data Links servers are involved in the transaction. When the application issues a COMMIT for a transaction, the database manager commits the transaction by using two-phase commit protocol. In the first phase, the database manager writes a PREPARE log record and distributes a PREPARE request to all Data Links servers. Each Data Links server then responds with one of the following:

- YES; signifying that the Data Links server is prepared to commit
- NO; because of an error, Data Links server is not prepared to commit.

The first phase is considered successful if all Data Links servers respond "YES".

The processing in the second phase depends on the outcome of the first phase. If at least one of the Data Links servers responded "NO", the database manager distributes ABORT requests to all the Data Links servers involved. The transaction is rolled back and the error message SQL0903N with reason code "03" is returned to the application. Otherwise, the database manager proceeds to commit the transaction as it usually does in the absence of involvement of Data Links servers. At the end of this processing, it distributes a COMMIT request to all the Data Links servers involved in the transaction.

If a failure occurs on a Data Links server leaving some transactions in the PREPARED state, these transactions are called *indoubt transactions*. The database manager is responsible for tracking the outcome of these transactions and eventually resolving them on the Data Links server. Whenever the database manager determines that a failure has potentially created indoubt transactions on a Data Links server, it marks the state of the Data Links server as needing crash recovery. It disallows any SQL requests involving the Data Links server while it is in this state. SQL0357N with reason code "03" is returned to the application which made the SQL request.

At the time of RESTART, ACTIVATE DATABASE, or first CONNECT processing, the database manager attempts to connect to each configured Data Links server and attempts to resolve its indoubt transactions by aborting or committing them. A Data Links server's state is marked as available if all of its indoubt transactions are resolved except those transactions that are also indoubt on the database manager. In the available state, SQL requests

involving the Data Links server are allowed. At the end of this attempt to resolve indoubt transactions if the database manager determines that a Data Links server still potentially has indoubt transactions which need resolution, it marks the state of the Data Links server as needing crash recovery. This can happen, for instance, if a Data Links server is not available during RESTART, ACTIVATE DATABASE, or first CONNECT processing; or, the Data Links server encounters a failure during that processing.

While a Data Links server configured to a database is in a state needing crash recovery, the database manager disallows SQL requests involving that particular Data Links server. SQL requests involving other data in the database are still allowed. The database manager starts a process which asynchronously attempts to complete crash recovery on each Data Links server requiring recovery. When the process successfully completes the crash recovery, the state of the Data Links server is marked as available thereby allowing further SQL requests involving it.

Backup Utility Considerations

DB2 ensures that by the time a backup operation completes, linked files at Data Links servers running the DB2 Data Links Manager are also backed up. (The backup utility can run either online or offline, and the backup image can be of either a database or a table space.) The description that follows only applies to files that are linked by DATALINK columns that have the RECOVERY parameter set to YES. (Files that are referenced by DATALINK columns for which RECOVERY=NO is specified are not backed up.)

When files are linked, the Data Links servers schedule them to be copied asynchronously to an archive server such as TSM, or to disk. When the backup utility runs, DB2 ensures that all files scheduled for copying have been copied. At the beginning of backup processing, DB2 contacts all Data Links servers that are specified in the DB2 configuration file. If one or more Data Links servers are configured for the database, the backup operation will succeed, even if a Data Links server is not available. When the Data Links server restarts, backup processing will be completed on that Data Links server before it becomes available to the database again. If a Data Links server has one or more linked files and is not running, or stops running during the backup operation, the backup image will not contain complete DATALINK information. The backup operation will complete successfully. The comment field in the history file for that backup operation will identify the DLM server that failed or, if there are more than four DLM servers that failed, the number of DLM servers that failed. Once the DLMs have been successfully backed up, the comment field is reset to its previous value.

Before the Data Links server can be marked as available to the database again, backup processing for all outstanding backups must complete successfully. (This is done automatically by an asynchronous process.) If a backup

operation is initiated when there are already twice the value of *num_db_backups* (see below) outstanding backups waiting to be completed on the Data Links server, the backup operation will fail. That Data Links server must be restarted, and the outstanding backups completed before additional backups are allowed.

When a file is unlinked, it is either deleted or returned to its previous permissions, depending on the value specified for the ON UNLINK parameter. A successful backup operation can cause the Data Links servers to clean up the archived versions of files on the archive server (either disk or TSM; see "Garbage Collection" on page 50). The *num_db_backups* database configuration parameter specifies the number of DB2 database backups before archived versions of the files (that were unlinked) are removed. For more information about this configuration parameter, see the *Administration Guide: Performance* book.

When unlinked files are removed, the information about the unlinked files is also removed from the Data Links server registration tables.

Choosing a Backup Method for DB2 Data Links Manager on AIX

In addition to Disk Copy and XBSA, you can also use Tivoli Storage Manager (TSM) for backing up files that reside on a Data Links server.

To use Tivoli Storage Manager as an archive server:

1. Install Tivoli Storage Manager on the Data Links server. For more information, see your Tivoli Storage Manager product documentation.
2. Register the Data Links server client application with the Tivoli Storage Manager server. For more information, see your Tivoli Storage Manager product documentation.
3. Add the following environment variables to the Data Links Manager Administrator's `db2profile` or `db2cshrc` script files:

```
(for Bash, Bourne, or Korn shell)
export DSMI_DIR=/usr/tivoli/tsm/client/api/bin
export DSMI_CONFIG=$HOME/tsm/dsm.opt
export DSMI_LOG=$HOME/dldump
export PATH=$PATH:$DSMI_DIR

(for C shell)
setenv DSMI_DIR /usr/tivoli/tsm/client/api/bin
setenv DSMI_CONFIG ${HOME}/tsm/dsm.opt
setenv DSMI_LOG ${HOME}/dldump
setenv PATH=${PATH}:$DSMI_DIR
```

4. Ensure that the `dsm.sys` TSM system options file is located in the `$DSMI_DIR` directory.

5. Ensure that the `dsm.opt` TSM user options file is located in the *INSTHOME*`/tsm` directory, where *INSTHOME* is the home directory of the Data Links Manager Administrator.
6. Set the *PASSWORDACCESS* option to `generate` in the `/usr/tivoli/tsm/client/api/bin/dsm.sys` Tivoli Storage Manager system options file.
7. Register TSM password with the `generate` option *before* starting the Data Links File Manager for the first time. This way, you will not need to provide a password when the Data Links File Manager initiates a connection to the TSM server. For more information, see your TSM product documentation.
8. Set the DLFM_BACKUP_TARGET registry variable to TSM. The value of DLFM_BACKUP_DIR_NAME registry variable will be ignored in this case. This will activate the Tivoli Storage Manager backup option.

   Notes:

   a. If you change the setting of the DLFM_BACKUP_TARGET registry variable between TSM and disk at run time, you should be aware that the archived files are not moved to the newly specified archive location. For example, if you start the Data Links File Manager with the DLFM_BACKUP_TARGET registry value set to TSM, and change the registry value to a disk location, all newly archived files will be stored in the new location on the disk. The files that were previously archived to TSM will not be moved to the new disk location.

   b. To override the default TSM management class there is a new registry variable called DLFM_TSM_MGMTCLASS. If this registry variable is left unset then the default TSM management class will be used.
9. Stop the Data Links File Manager by entering the **dlfm stop** command.
10. Start the Data Links File Manager by entering the **dlfm start** command.

Choosing a Backup Method for DB2 Data Links Manager in the Solaris Operating Environment

In addition to Disk Copy and XBSA, you can also use Tivoli Storage Manager (TSM) for backing up files that reside on a Data Links server.

To use Tivoli Storage Manager as an archive server:

1. Install Tivoli Storage Manager on the Data Links server. For more information, see your Tivoli Storage Manager product documentation.
2. Register the Data Links server client application with the Tivoli Storage Manager server. For more information, see your Tivoli Storage Manager product documentation.
3. Add the following environment variables to the Data Links Manager Administrator's `db2profile` or `db2cshrc` script files:

```
(for Bash, Bourne, or Korn shell)
export DSMI_DIR=/opt/tivoli/tsm/client/api/bin
export DSMI_CONFIG=$HOME/tsm/dsm.opt
export DSMI_LOG=$HOME/dldump
export PATH=$PATH:/opt/tivoli/tsm/client/api/bin

(for C shell)
setenv DSMI_DIR /opt/tivoli/tsm/client/api/bin
setenv DSMI_CONFIG ${HOME}/tsm/dsm.opt
setenv DSMI_LOG ${HOME}/dldump
setenv PATH=${PATH}:/opt/tivoli/tsm/client/api/bin
```

4. Ensure that the `dsm.sys` TSM system options file is located in the `/opt/tivoli/tsm/client/api/bin` directory.
5. Ensure that the `dsm.opt` TSM user options file is located in the *INSTHOME*`/tsm` directory, where *INSTHOME* is the home directory of the Data Links Manager Administrator.
6. Set the *PASSWORDACCESS* option to `generate` in the `/opt/tivoli/tsm/client/api/bin/dsm.sys` Tivoli Storage Manager system options file.
7. Register TSM password with the `generate` option *before* starting the Data Links File Manager for the first time. This way, you will not need to provide a password when the Data Links File Manager initiates a connection to the TSM server. For more information, see your TSM product documentation.
8. Set the DLFM_BACKUP_TARGET registry variable to TSM. The value of DLFM_BACKUP_DIR_NAME registry variable will be ignored in this case. This will activate the Tivoli Storage Manager backup option.

   Notes:

   a. If you change the setting of the DLFM_BACKUP_TARGET registry variable between TSM and disk at run time, you should be aware that the archived files are not moved to the newly specified archive location. For example, if you start the Data Links File Manager with the DLFM_BACKUP_TARGET registry value set to TSM, and change the registry value to a disk location, all newly archived files will be stored in the new location on the disk. The files that were previously archived to TSM will not be moved to the new disk location.

   b. To override the default TSM management class there is a new registry variable called DLFM_TSM_MGMTCLASS. If this registry variable is left unset then the default TSM management class will be used.
9. Stop the Data Links File Manager by entering the **dlfm stop** command.
10. Start the Data Links File Manager by entering the **dlfm start** command.

Choosing a Backup Method for DB2 Data Links Manager on Windows NT

Whenever a DATALINK value is inserted into a table with a DATALINK column that is defined for recovery, the corresponding DATALINK files on the

Data Links server are scheduled to be backed up to an archive server. Currently, Disk Copy (default method) and Tivoli Storage Manager are the two options that are supported for file backup to an archive server. Future releases of DB2 Data Links Manager for Windows NT will support other vendors' backup media and software.

Disk Copy (default method)

When the backup utility is invoked on the DB2 server, it ensures that the linked files in the database are backed up on the Data Links server to the directory specified by the DLFM_BACKUP_DIR_NAME environment variable. The default value for this variable is `c:\dlfmbackup`, where `c:\` represents the Data Links Manager backup installation drive.

To set this variable to c:\dlfmbackup, enter the following command:

```
db2set -g DLFM_BACKUP_DIR_NAME=c:\dlfmbackup
```

The location specified by the DLFM_BACKUP_DIR_NAME environment variable must *not* located on a file system using a Data Links Filesystem Filter and that the required space is available in the directory you specified for the backup files.

Also, ensure that the DLFM_BACKUP_TARGET variable is set to LOCAL by entering the following command:

```
db2set -g DLFM_BACKUP_TARGET=LOCAL
```

After setting or changing these variables, stop and restart the Data Links File Manager using the **dlfm stop** and **dlfm start** commands.

Tivoli Storage Manager

To use Tivoli Storage Manager as an archive server:

1. Install Tivoli Storage Manager on the Data Links server. For more information, see your Tivoli Storage Manager product documentation.
2. Register the Data Links server client application with the Tivoli Storage Manager server. For more information, see your Tivoli Storage Manager product documentation.
3. Click on **Start** and select **Settings —> Control Panel —> System**. The System Properties window opens. Select the **Environment** tab and enter the following environment variables and corresponding values:

Variable	Value
DSMI_DIR	c:\tsm\baclient
DSMI_CONFIG	c:\tsm\baclient\dsm.opt

Variable	Value
DSMI_LOG	c:\tsm\dldump

4. Ensure that the `dsm.sys` TSM system options file is located in the `c:\tsm\baclient` directory.
5. Ensure that the `dsm.opt` TSM user options file is located in the `c:\tsm\baclient` directory.
6. Set the *PASSWORDACCESS* option to `generate` in the `c:\tsm\baclient\dsm.sys` Tivoli Storage Manager system options file.
7. Register TSM password with the `generate` option *before* starting the Data Links File Manager for the first time. This way, you will not need to provide a password when the Data Links File Manager initiates a connection to the TSM server. For more information, see your TSM product documentation.
8. Set the DLFM_BACKUP_TARGET environment variable to TSM using the following command:

   ```
   db2set -g DLFM_BACKUP_TARGET=TSM
   ```

   The value of the DLFM_BACKUP_DIR_NAME environment variable will be ignored in this case. This will activate the Tivoli Storage Manager backup option.

   Notes:

   a. If you change the setting of the DLFM_BACKUP_TARGET environment variable between TSM and LOCAL at run time, you should be aware that the archived files are not moved to the newly specified archive location. For example, if you start the Data Links File Manager with the DLFM_BACKUP_TARGET environment variable set to TSM, and change its value to LOCAL, all newly archived files will be stored in the new location on the disk. The files that were previously archived to TSM will not be moved to the new disk location.

   b. To override the default TSM management class there is a new environment variable called DLFM_TSM_MGMTCLASS. If this variable is left unset then the default TSM management class will be used.
9. Stop the Data Links File Manager by entering the **dlfm stop** command.
10. Start the Data Links File Manager by entering the **dlfm start** command.

Backing up a Journaled File System on AIX

You can perform an offline backup of a journaled file system on AIX after stopping the Data Links Manager. The following approach, which removes the requirement of stopping the Data Links Manager, is suggested for users who require higher availability.

1. Access the CLI source file `quiesce.c` and the shell script `online.sh`. These files are located in the `/samples/dlfm` directory.
2. Compile `quiesce.c`:

   ```
   xlC -o quiesce  -L$HOME/sqllib/lib -I$HOME/sqllib/include -c quiesce.c
   ```

3. As root, run the script on the node that has the DLFS file system.

The shell script `online.sh` assumes that you have a catalog entry on the Data Link Manager node for each database that is registered with the Data Link Manager. It also assumes that `/etc/filesystems` has the complete entry for the DLFS file system. The shell script does the following:

- Quiesces all the tables in databases that are registered with the Data Links Manager. This will stop any new activity.
- Unmounts and remounts the file system as a read-only file system.
- Performs a file system backup.
- Unmounts and remounts the file system as a read-write file system.
- Resets the DB2 tables; that is, brings them out of the quiesce state.

The script must be modified to suit your environment as follows:

1. Select the backup command and put in the do_backup function of the script.
2. Set the following environment variables within the script:
   - DLFM_INST: set this to the DLFM instance name.
   - PATH_OF_EXEC: set this to the path where the "quiesce" executable resides.

Invoke the script as follows:

```
online.sh <filesystem_name>
```

Restore and Rollforward Utility Considerations

The information that follows applies if you have a DATALINK column (or columns) that is defined with RECOVERY=YES option for a table. If a table has a DATALINK column defined with the RECOVERY=NO option, the table is put in datalink reconcile pending state at the end of the restore operation. For more information, see "Reconciling Data Links" on page 74.

During restore operations, tables with DATALINK columns may be put into one of two states:

- *Datalink reconcile not possible*

When a table is in datalink reconcile not possible state, it is available for unrestricted actions against columns that are not DATALINK columns. When a DATALINK column is involved in a SELECT statement, a warning is returned. You can issue UPDATE calls to DATALINK columns (with some restrictions: see "Removing a Table From Datalink Reconcile Not Possible State" on page 74 for details). You cannot issue INSERT and DELETE statements, because they involve the DATALINK column.

- *Datalink reconcile pending*

  When a table is in datalink reconcile pending> state, it is available for unrestricted actions against columns that are not DATALINK columns. When a DATALINK column is involved in a SELECT statement, a warning is returned. You cannot issue any DML statements such as UPDATE, INSERT, or DELETE.

These states are reported in the `db2diag.log` file when the restore or rollforward utilities run. You can also use the **db2dart** command to obtain this information.

When you restore a database or table space, the following conditions must be satisfied for the restore operation to succeed:

- If any Data Links Server recorded in the backup file is not running, the restore operation will still complete successfully.

  Tables with DATALINK column information that are affected by the missing Data Links server will be put in datalink reconcile pending state after the restore operation (or the rollforward operation, if used) completes. Before the Data Links servers can be marked as available to the database again, this restore processing must complete successfully. The asynchronous process that completes backup processing once the DLM becomes available (see "Backup Utility Considerations" on page 57 also completes restore processing.
- If any Data Links Server recorded in the backup file stops running during the restore operation, the restore operation will fail. The restore can be restarted with the Data Links Server down (see above).
- If a previous database restore operation is still incomplete on any Data Links server, subsequent database or table space restore operations will fail until those Data Links servers are restarted, and the incomplete restore is completed.
- Information about all DATALINK columns that are recorded in the backup file must exist in the appropriate Data Links servers' registration tables.

  If all the information about the DATALINK columns is not recorded in the registration tables, the table with the missing DATALINK column information is put in datalink reconcile not possible state after the restore operation (or the rollforward operation, if used) completes.

If the backup is not recorded in the registration tables, it may mean that the backup file that is provided is earlier than the value for num_db_backups and has already been ″garbage collected″. This means that the archived files from this earlier backup have been removed and cannot be restored. All tables that have DATALINK columns are put in datalink reconcile pending state.

If the backup is not recorded in the registration tables, it may mean that backup processing has not yet been completed because the Data Links server is not running. All tables that have DATALINK columns are put in datalink reconcile pending state. When the Data Links server is restarted, backup processing will be completed before restore processing.

The table remains available to users, but the values in the DATALINK columns may not reference the files accurately (for example, a file may not be found that matches a value for the DATALINK column). If you do not want this behavior, you can put the table into check pending state by issuing the ″SET CONSTRAINTS for tablename TO DATALINK RECONCILE PENDING″ statement.

If, after a restore operation, you have a table in datalink reconcile not possible state, you can fix the DATALINK column data in one of the ways suggested under “Removing a Table From Datalink Reconcile Not Possible State” on page 74.

Note: In the process of marking a file from the unlinked state to the linked state, that file may have to be retrieved from an archive server to the file system. If an error occurs during this process (for example, a file cannot be copied into the file system because of duplicate file names), the corresponding table is placed in datalink reconcile pending state.

It is strongly recommended that the `datalink.cfg` file be archived to cover certain unusual recovery cases, since the `datalink.cfg` file in the database backup image only reflects the `datalink.cfg` as of the backup time. Having the latest `datalink.cfg` file is required to cover all recovery cases. Therefore, the `datalink.cfg` file must be backed up after every ADD DATALINKS MANAGER or DROP DATALINKS MANAGER command invocation. This would help to retrieve the latest `datalink.cfg` file, if the latest `datalink.cfg` file is not available on disk.

If the latest `datalink.cfg` file is not available on disk, replace the existing `datalink.cfg` file (restored from a backup image) with the latest `datalink.cfg` file that was archived before running a rollforward operation. Do this after the database is restored.

Restoring Databases From an Offline Backup Without Rolling Forward

You can only restore without rolling forward at the database level, not the table space level. To restore a database without rolling forward, you can either

restore a non-recoverable database (that is, a database that uses circular logging), or specify the WITHOUT ROLLING FORWARD parameter on the RESTORE DATABASE command.

If you use the restore utility with the WITHOUT DATALINK option, all tables with DATALINK columns are placed in datalink reconcile pending (DRP) state, and no reconciliation is performed with the Data Links servers during the restore operation.

If you do not use the WITHOUT DATALINK option, and a Data Links server recorded in the backup file is no longer defined to the database (that is, it has been dropped using the DROP DATALINKS MANAGER command), tables that contain DATALINK data referencing the dropped Data Links server are put in DRP state by the restore utility.

If you do not use the WITHOUT DATALINK option, all the Data Links servers are available, and all information about the DATALINK columns is fully recorded in the registration tables, the following occurs for each Data Links server recorded in the backup file:

- All files linked after the backup image that was used for the database restore operation are marked as unlinked (because they are not recorded in the backup image as being linked).
- All files that were unlinked after the backup image, but that were linked before the backup image was taken, are marked as linked (because they are recorded in the backup image as being linked). If the file was subsequently linked to another table in another database, the restored table is put into the datalink reconcile pending state.

Note: The above cannot be done if the backup image that was used for the database restore operation was taken when at least one Data Links server was not running, since the DATALINK information in the backup is incomplete. The above is also not done if the backup image that was used for the database restore operation was taken after a database restore operation with or without rollforward recovery. In both cases, all tables with DATALINK columns are placed in datalink reconcile pending state, and no reconciliation is performed with the Data Links servers during the restore operation.

Restoring Databases and Table Spaces, and Rolling Forward to the End of the Logs

If you restore, then roll the database or table spaces forward to the end of logs (meaning that all logs are provided), a reconciliation check is not required unless at least one of the Data Links servers recorded in the backup file is not running during the restore operation. If you are not sure whether all the logs were provided for the rollforward operation, or think that you may need to reconcile DATALINK values, do the following:

1. Issue the SQL statement for the table (or tables) involved:

   ```
   SET CONSTRAINTS FOR tablename TO DATALINK RECONCILE PENDING
   ```

   This puts the table in both datalink reconcile pending state and check pending state.
2. If you do not want a table to be in check pending state, issue the following SQL statement:

   ```
   SET CONSTRAINTS FOR tablename IMMEDIATE CHECKED
   ```

   This takes the table out of check pending state, but leaves it in datalink reconcile pending state. You must use the reconcile utility to take the table out of this state.

It may happen that the backup file contains DATALINK data that refers to a DB2 Data Links Manager (that is, a DB2 Data Links Manager was registered to the database when the backup was taken) that has been dropped from the database. For each table space being rolled forward that contains at least one table with DATALINK data referencing the dropped DB2 Data Links Manager, all tables with DATALINK columns are put in DRP state by the rollforward utility.

Restoring Databases and Table Spaces, and Rolling Forward to a Point in Time

When working with Data Links tables, you can roll forward to the end of the logs or to a specified point in time.

Tables in table spaces that are rolled forward to a point in time are placed in datalink reconcile pending state at the end of the rollforward operation. You should use the reconcile utility to remove them from this state. For more information, see “Reconciling Data Links” on page 74.

Point in Time Rollforward Example

Following is a simple scenario showing the files that need to be retained in order to handle backup and recovery. The example shows changes to the value of a single row in column of type DATALINK together with the files that the DB2 Data Links Manager needs to retain to support recovery. For this example, the assumption is made that there is no support for point in time recovery of these files earlier than the last backup. Data Links servers running the DB2 Data Links Manager do not have such a restriction. Observe that fileA exists until time 3, at which time it is deleted because it was unlinked at time 2, and the policy for the database in this example is to keep the unlinked files until the next backup is run (that is, the *num_db_backups* database configuration parameter is set to 1).

Time	1	2	3	4	5	6	7
Activity	Create	Update	Backup	Update	Update	Delete	Restore to 5
Column Value	valueA	valueB	valueB	valueC	valueD	-	valueD
Linked File	fileA	fileB	fileB	fileC	fileD	-	fileD
Extra Files Kept by Data Links File Manager		fileA		fileB	fileB, fileC	fileB, fileC, fileD	fileB, fileC

Note: Recovery of linked files is always done in conjunction with the rest of the database.

DB2 Data Links Manager and Recovery Interactions

The following table shows the different types of recovery that you can perform, the DB2 Data Links Manager processing that occurs during restore and rollforward processing, and whether you need to invoke the reconcile utility after the recovery is complete:

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Non-recoverable database			
Database restore of a complete backup, all Data Links Servers up	Fast reconcile is performed	N/A	Can be optionally run if problem with file links is suspected
Database restore using WITHOUT DATALINK option	Tables put in datalink reconcile pending state	N/A	Required

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore of a complete backup, at least one Data Links server down	Fast reconcile is performed only on those tables in table spaces that do not have links to a Data Links server that is down, other tables put in datalink reconcile pending state	NA	Required for tables in table spaces with links to the Data Links server that is down
Database restore of an incomplete backup, all Data Links servers up	Fast reconcile is not performed, all tables with DATALINK columns put in datalink reconcile pending state	NA	Required
Recoverable database			
Database restore using WITHOUT ROLLING FORWARD option, using a complete backup, all Data Links servers up	Fast reconcile is performed	N/A	Optional
Database restore using WITHOUT ROLLING FORWARD and WITHOUT DATALINK options, using a complete or incomplete backup, Data Links servers up or down	Tables put in datalink reconcile pending state	N/A	Required

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore using WITHOUT ROLLING FORWARD option, using a complete backup, at least one Data Links server down	Fast reconcile is performed only on those tables in table spaces that do not have links to the Data Links servers that are down, other tables put in datalink reconcile pending state	N/A	Required on tables in table spaces with links to the Data Links servers that are down
Database restore using WITHOUT ROLLING FORWARD option, using an incomplete backup, Data Links servers up or down	Fast reconcile is not performed, all tables with DATALINK columns put in datalink reconcile pending state	N/A	Required
Database restore and rollforward to end of logs, using a complete backup, all Data Links servers up	No action	No action	Optional
Database restore and rollforward to end of logs, using a complete backup, at least one Data Links server down during rollforward processing	No action	No action	Optional
Database restore and rollforward to end of logs, using a complete or an incomplete backup, any Data Links server down during restore	No action	All tables with DATALINK columns put in datalink reconcile pending state	Required for all tables with DATALINK columns

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore and rollforward to end of logs, using an incomplete backup, all Data Links servers up during restore	No action	No action	Optional
Database restore and rollforward to end of logs, using a complete or an incomplete backup, all Data Links servers up, backup unknown at any Data Links server	No action	All tables in table spaces with links to a Data Links server where the backup is unknown put in datalink reconcile pending state	Required
Table space restore and rollforward to end of logs, using a complete backup, all Data Links servers up	No action	No action	Optional
Table space restore and rollforward to end of logs, using a complete backup, at least one Data Links server down during rollforward processing	No action	No action	Optional
Table space restore and rollforward to end of logs, using a complete or an incomplete backup, any Data Links server down during restore processing	No action	All tables in table spaces with links to any Data Links server that is down put in datalink reconcile pending state	Required for tables in table spaces with links to any Data Links server that is down

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Table space restore and rollforward to end of logs, using an incomplete backup, all Data Links servers up	No action	No action	Optional
Database restore and rollforward to a point in time, using a complete or an incomplete backup, Data Links servers up or down during restore and/or rollforward processing	No action	Tables put in datalink reconcile pending state	Required
Table space restore and rollforward to a point in time, using a complete or an incomplete backup, Data Links servers up or down during restore and/or rollfoward processing	No action	Tables put in datalink reconcile pending state	Required
Database restore to a different database name, alias, hostname, or instance with no rollforward (see Note 1 on page 73)	Tables put in datalink reconcile not possible state	N/A	Optional, but tables in datalink reconcile not possible state must be manually fixed
Database restore to a different database name, alias, hostname or instance, and rollforward	No action	Tables put in datalink reconcile not possible state	Optional, but tables in datalink reconcile not possible state must be manually fixed

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore from an unusable backup (image has been garbage-collected on the Data Links server) with no rollforward (see Note 1), with or without the WITHOUT DATALINK option	Tables put in datalink reconcile pending state	No action	Required
Database restore from an unusable backup (image has been garbage-collected on the Data Links server), and rollforward, with or without WITHOUT DATALINK option	No action	Tables put in datalink reconcile pending state	Required
Table space restore from an unusable backup (image has been garbage-collected on the Data Links server), and rollforward	No action	Tables put in datalink reconcile pending state	Required

Notes:

1. A restore operation using an online backup image and the WITHOUT ROLLING FORWARD option, or a restore operation using an offline backup image.
2. A *complete* backup is a backup taken when all required Data Links servers were running. An *incomplete* backup is a backup taken when at least one required Data Links server was not running.
3. Fast reconcile processing will not be performed if the backup image that was used for the database restore operation was taken after a database restore, with or without rollforward. In this case, all tables with DATALINK columns are put in datalink reconcile pending state.

Removing a Table From Datalink Reconcile Not Possible State

A restored table (or tables) with a DATALINK column is put in datalink reconcile not possible state if a table space is restored from a backup that is earlier than the value specified for the *num_db_backups* database configuration parameter. For more information about this configuration parameter, see the *Administration Guide: Performance* book.

DB2 still allows the table to be accessed, even though the DATALINK column values may not be valid. If you want to prevent access to a table with possibly inconsistent DATALINK column values, issue the SET CONSTRAINTS for *tablename* TO DATALINK RECONCILE PENDING command. You can update the DATALINK values as follows:

- Using the SQL UPDATE statement, set the data location part of a DATALINK column value to a zero-length URL if the column is not nullable, or to NULL if the column is nullable.
- Restore the files on the appropriate Data Links servers. Then run an application that issues SELECT statements to read the DATALINK column values, and issues UPDATE statements to update the DATALINK column with the same values. Note that the datalink reconcile not possible state must be in effect while the DATALINK column values are being updated. After the update operation completes, the files will be marked as linked on the appropriate Data Links servers.

You then reset the datalink reconcile not possible state by issuing the following command:

```
SET CONSTRAINTS FOR tablename DATALINK RECONCILE PENDING IMMEDIATE UNCHECKED
```

Reconciling Data Links

You use the reconcile utility to reconcile data links. The utility is initiated from DB2, and involves all the Data Links servers running the DB2 Data Links Manager that are referenced by the DATALINK column values. It validates that the referenced files either exist on the Data Links server, or that links can be re-established. The following sections describe how DB2 detects whether you need to reconcile data links, and how to reconcile them.

If a Data Links server file reference does not exist or cannot be re-established, the reconcile utility places a copy of the rows in error along with a reason for each into an exception table (if specified), then modifies the offending rows. If the exception table is not specified, the DATALINK column values for which a file reference could not be re-established are copied to an exception report file along with a column-ID and reason. You can use the exception table (if specified) information or the report to update the rows to make the required corrections. The exception table used with the reconcile utility is identical to the exception table used by the load utility. For more information about the load utility, see the *Data Movement Utilities Guide and Reference*. The report uses

the naming convention *report*.exp (the .exp extension is supplied by the reconcile utility). For example, you can invoke the reconcile utility with the following statement:

```
db2 RECONCILE dept DLREPORT /u/scottba/report FOR EXCEPTION excptab
```

This command reconciles the table called dept, and writes exceptions to the exception table excptab, which was created by the user. Information about files that were unlinked during reconciliation are written to the file report.ulk, which is created in the directory /u/scottba. If *FOR EXCEPTION excptab* is not specified, then the exception information is written to the file report.exp, which is created in the directory /u/scottba. For more information about the reconcile utility, see the *Command Reference*.

Detection of Situations That Require Reconciliation

Following are some situations in which you may need to run the reconcile utility:

- The entire database is restored and rolled forward to a point in time. Because the entire database is rolled forward to a committed transaction, no tables will be in check pending state (due to referential constraints or check constraints). All data in the database is brought to a consistent state. The DATALINK columns, however, may not be synchronized with the metadata in the DB2 Data Links Manager, and reconciliation is required.

  In this situation, tables with DATALINK data will already be in DRP state. You should invoke the reconcile utility for each of these tables.
- A particular Data Links server running the DB2 Data Links Manager loses track of its metadata. This can occur for different reasons. For example:
  - The Data Links server was cold started.
  - The Data Links server metadata was restored to a back-level state.

  In some situations, such as during SQL UPDATEs and DELETEs, DB2 may be able to detect a problem with the metadata in a Data Links server. In these situations, the SQL statement would fail. You would put the table in DRP state by using the SET CONSTRAINTS statement, then run the reconcile utility on that table.
- A file system is not available (for example, because of a disk crash) and is not restored to the current state. In this situation, files may be missing.
- A DB2 Data Links Manager is dropped from a database, and there are DATALINK FILE LINK CONTROL values referencing that DB2 Data Links Manager. You should run the reconcile utility on such tables.

Summary of Procedure for Reconciliation

If you need to reconcile data links because of point in time recovery, or because Data Links servers running the DB2 Data Links Manager and DB2 control information do not match:

1. Put the table in datalink reconcile pending state by issuing the SET CONSTRAINTS statement. (In some situations, DB2 will do this for you.)
2. Use the reconcile utility to resolve the links, and take the appropriate actions for the exceptions in the exception table or in the exception report.

Chapter 2. Database Backup

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

The following topics are covered:

Backup Overview

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

```
db2 backup db sample
```

If the command completes successfully, you will have acquired a new backup image that is located in the path or the directory from which the command was issued. It is located in this directory because the command in this example does not explicitly specify a target location for the backup image. On Windows NT/2000, for example, this command (when issued from the root directory) creates an image that appears in a directory listing as follows:

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

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

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

- A directory (for backups to disk or diskette)
- A device (for backups to tape)
- A Tivoli Storage Manager (TSM) server (see "Appendix G. Tivoli Storage Manager" on page 433)
- Another vendor's server
- Specified through a user exit program (OS/2 only).

The recovery history file is updated automatically with summary information whenever you invoke a full database backup operation. This file is created in the same directory as the database configuration file. For more information about the recovery history file, see "Understanding the Recovery History File" on page 48.

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

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

For example:

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

On other platforms, a four-level subdirectory tree is used:

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

For example (Windows NT/2000):

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

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

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

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

Node number — The node number. In non-partitioned database systems, this is always `NODE0000`. In partitioned database systems, it is `NODExxxx`,

where *xxxx* is the number assigned to the node in the `db2nodes.cfg` file.

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

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

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

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

When a backup image is written to tape:

- File names are not created, but the information described above is stored in the backup header for verification purposes.
- A tape device must be available through the standard operating system interface. On a large partitioned database system, however, it may not be practical to have a tape device dedicated to each database partition server. You can connect the tape devices to one or more TSM servers, so that access to these tape devices is provided to each database partition server. For more information about TSM, see "Appendix G. Tivoli Storage Manager" on page 433.
- On a partitioned database system, you can also use products that provide virtual tape device functions, such as REELlibrarian 4.2 or CLIO/S. You can use these products to access the tape device connected to other nodes (database partition servers) through a pseudo tape device. Access to the remote tape device is provided transparently, and the pseudo tape device can be accessed through the standard operating system interface.

Privileges, Authorities, and Authorization Required to Use Backup

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

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

Using Backup

Before Using Backup

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

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

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

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

If a database was created with a previous release of the database manager, and the database has not been migrated, you must migrate the database before you can back it up. For information about migrating databases, see the *Administration Guide: Planning* book.

Invoking Backup

The backup utility can be invoked through:

- The command line processor (CLP).

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

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

- The Backup Database notebook or wizard in the Control Center. To open the Backup Database notebook or wizard:

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

  For general information about the Control Center, see the *Administration Guide*. Detailed information is provided through the online help facility within the Control Center.

- An application programming interface (API), **sqlubkp**. For information about this API, see "Backup Database API" on page 88. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*.

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.

For detailed information about this utility, see "db2ckbkp - Check Backup" on page 306.

Backing Up to Tape

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

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

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

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

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

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

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

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

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

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

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

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

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

Notes:

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

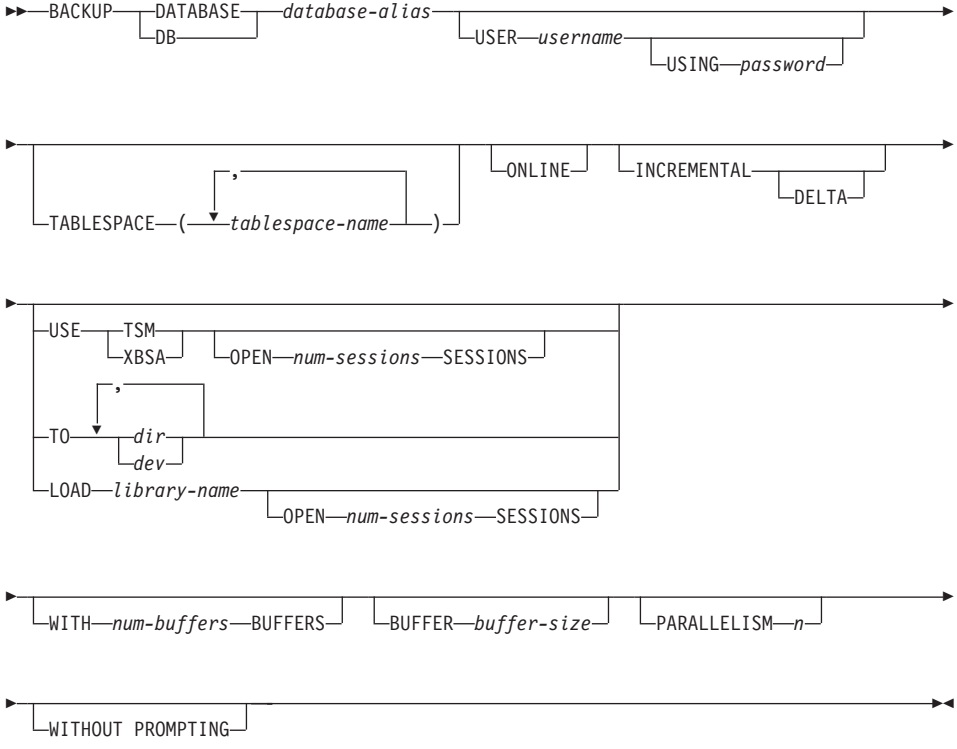
Backing Up to Named Pipes

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

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

BACKUP DATABASE Command

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.

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

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

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

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

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

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

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

TO dir/dev
A list of directory or tape device names. The full path on which the directory resides must be specified. The target must reside on the database server. This parameter may be repeated to specify the target directories and devices that the backup image will span. If more than one target is specified (target1, target2, and target3, for example), target1 will be opened first. The media header and special files (including the configuration file, table space table, and history file) are placed in target1. All remaining targets are opened, and are then used in parallel during the backup operation. Because there is no general tape support on OS/2 or the Windows operating system, each type of tape device requires a unique device driver. To back up to the FAT file system on OS/2 or the Windows operating system, users must conform to the 8.3 naming restriction.

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

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

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

t Terminate. Abort the backup operation.

Tape is not supported on OS/2. On OS/2, `0` or `0:` can be specified to cause the backup operation to call a user exit program (see "Appendix H. User Exit for Database Recovery" on page 439). The database is quiesced before an online database backup operation with a user exit program starts. The backup utility waits until all transactions are committed or rolled back. While the utility is running, all new transactions wait until the backup operation completes.

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 OS/2 or the Windows operating system) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path on which the user exit program resides.

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

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

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

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

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

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

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

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

WITHOUT PROMPTING
Specifies that the backup will run unattended, and that any actions which normally require user intervention will return an error message.

Backup Database API

C API Syntax

```
/* File: sqlutil.h */
/* API: Backup Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlubkp (
    char * pDbAlias,
    sqluint32     BufferSize,
    sqluint32     BackupMode,
    sqluint32     BackupType,
    sqluint32     CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    sqluint32     NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaTargetList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    sqluint32 VendorOptionsSize,
    void * pVendorOptions,
    sqluint32 Parallelism,
    sqluint32 * pBackupSize,
    void * pReserved4,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Backup Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbkp (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    unsigned short * pReserved1,
    char * pDbAlias,
    sqluint32 BufferSize,
    sqluint32 BackupMode,
    sqluint32 BackupType,
    sqluint32 CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    sqluint32 NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaTargetList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    sqluint32 VendorOptionsSize,
    void * pVendorOptions,
    sqluint32 Parallelism,
    sqluint32 * pBackupSize,
    void * pReserved4,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

UserNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

pReserved1.
Reserved for future use.

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

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

BackupMode
Input. Specifies the backup mode. Valid values (defined in `sqlutil`) are:

SQLUB_OFFLINE
Offline gives an exclusive connection to the database.

SQLUB_ONLINE
Online allows database access by other applications while the backup operation is in progress.

Note: An online backup operation may time out if there is an IX lock on `sysibm.systables`, because the DB2 backup utility acquires S locks on SMS LOB objects and IN locks on all other objects.

BackupType
Input. Specifies the type of backup to be taken. Valid values (defined in `sqlutil`) are:

SQLUB_FULL
Specifies a full (non-incremental) database backup operation. This value will not be supported in the future. Use `SQLUB_DB` to specify a full database backup operation.

SQLUB_DB
Specifies a backup of all table spaces in the database.

SQLUB_TABLESPACE
Specifies a table space-level backup operation. Provide a list of table spaces to be backed up in the *pTablespaceList* parameter.

SQLUB_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.

SQLUB_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.

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

SQLUB_BACKUP
Start the backup operation.

SQLUB_NOINTERRUPT
Start the backup operation. Specifies that the backup operation will run unattended. 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 backup operation have been mounted, and utility prompts are not desired.

SQLUB_CONTINUE
Continue the backup operation after the user has performed some action requested by the utility (continue after mounting a new tape, for example).

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

SQLUB_DEVICE_TERMINATE
Remove a particular device from the list of devices being used by the backup utility. When a particular medium is full, the backup utility returns a warning to the caller (while continuing to process using the remaining devices). Invoke the backup utility again with this caller action to remove the device that generated the warning from the list of devices being used.

SQLUB_PARM_CHECK
Used to validate parameters without performing a backup operation. 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.

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

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

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

NumBuffers
Input. Specifies the number of backup buffers to be used.

pTablespaceList
Input. List of table spaces to be backed up. Required for table space-level backup operations only. See "Data Structure: SQLU-TABLESPACE-BKRST-LIST" on page 100.

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

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks, or diskettes). Provide a list of *sqlu_media_entry* structures. On OS/2 or the Windows operating system, the entries can be directory paths only, not tape device names.

SQLU_TSM_MEDIA
TSM. If an *sqlu_media_entry* structure is not being used to specify a path for the backup image, initialize the *media* pointer in the *sqlu_media_list_targets* structure 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 an *sqlu_vendor* structure.

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

See "Data Structure: SQLU-MEDIA-LIST" on page 96.

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

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

pReserved2
Reserved for future use.

VendorOptionsSize
Input. The length of the *pVendorOptions* field which cannot exceed 65535 bytes.

pVendorOptions
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.

Parallelism
Input. Degree of parallelism (number of buffer manipulators).

pBackupSize
Output. Size of the backup image (in MB). Can be set to NULL.

pReserved4
Reserved for future use.

pReserved3
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

REXX API Syntax

```
BACKUP DATABASE dbalias USING :value [USER username USING password]

[TABLESPACE :tablespacenames] [ONLINE]

[LOAD vendor-library [OPTIONS vendor-options] [OPEN num-sessions SESSIONS] |
 TO :target-area |
 USE TSM [OPEN num-sessions SESSIONS] |
 USER_EXIT]

[ACTION caller-action] [WITH num-buffers BUFFERS] [BUFFERSIZE buffer-size]
[PARALLELISM parallelism-degree]
```

REXX API Parameters

dbalias
Alias of the database to be backed up.

value A compound REXX host variable to which the database backup information is returned. In the following, XXX represents the host variable name:

XXX.0 Number of elements in the variables (always 2)

XXX.1 The time stamp of the backup image

XXX.2 An application ID that identifies the agent that serves the application.

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

password
The password used to authenticate the user name.

tablespacenames
A compound REXX host variable containing a list of table spaces to be backed up. In the following, XXX is the name of the host variable:

XXX.0 Number of table spaces to be backed up

XXX.1 First table space name

XXX.2 Second table space name

XXX.3 and so on.

vendor-library
The name of the shared library (DLL on Windows operating systems or OS/2) containing the vendor backup and restore I/O functions to be used. It may contain the full path. If the full path is not given, defaults to the path on which the user exit program resides.

vendor-options
Information required by the vendor functions.

num-sessions
The number of I/O sessions to be used with TSM or the vendor product.

target-area
Local devices. Allows a combination of tapes, disks or diskettes. Provide a list in "Data Structure: SQLU-MEDIA-LIST" on page 96. On OS/2 or the Windows operating system, the entries can be directory paths only, not tape device names.

caller-action
Specifies action to be taken. Valid values are:

SQLUB_BACKUP
Start the backup operation.

SQLUB_NOINTERRUPT
Start the backup operation. Specifies that the backup operation will run unattended. 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 backup operation have been mounted, and utility prompts are not desired.

SQLUB_CONTINUE
Continue the backup operation after the user has performed some action requested by the utility (continue mounting a new tape, for example).

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

SQLUB_DEVICE_TERMINATE
Remove a particular device from the list of devices being used by the backup utility. When a particular medium is full, the backup utility returns a warning to the caller (while continuing to process using the remaining devices). Invoke the backup utility again with this caller action to remove the device that generated the warning from the list of devices being used.

SQLUB_PARM_CHECK
Used to validate parameters without performing a backup operation.

num-buffers
The number of backup buffers to be used.

buffer-size
Backup buffer size in 4-KB allocation units. The minimum value is 8 units.

parallelism-degree
Degree of parallelism (number of buffer manipulators).

Data Structure: SQLU-MEDIA-LIST

This structure is used to:

- Hold a list of *target* media for the backup image (see the "Backup Database API" on page 88).
- Hold a list of *source* media for the backup image (see the "Restore Database API" on page 116).
- Pass information to the DB2 load utility.

Table 1. Fields in the SQLU-MEDIA-LIST Structure

Field Name	Data Type	Description
MEDIA_TYPE	CHAR(1)	A character indicating media type.
SESSIONS	INTEGER	Indicates the number of elements in the array pointed to by the *target* field of this structure.
TARGET	Union	This field is a pointer to one of three types of structures. The type of structure pointed to is determined by the value of the *media_type* field. For more information on what to provide in this field, see the appropriate API.

Table 2. Fields in the SQLU-MEDIA-LIST-TARGETS Structure

Field Name	Data Type	Description
MEDIA	Pointer	A pointer to an *sqlu_media_entry* structure.
VENDOR	Pointer	A pointer to an *sqlu_vendor* structure.
LOCATION	Pointer	A pointer to an *sqlu_location_entry* structure.

Table 3. Fields in the SQLU-MEDIA-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the *media_entry* field. For languages other than C.
MEDIA_ENTRY	CHAR(215)	Path for a backup image used by the backup and restore utilities.

Table 4. Fields in the SQLU-VENDOR Structure

Field Name	Data Type	Description
RESERVE_LEN1	INTEGER	Length of the *shr_lib* field. For languages other than C.
SHR_LIB	CHAR(255)	Name of a shared library supplied by vendors for storing or retrieving data.
RESERVE_LEN2	INTEGER	Length of the *filename* field. For languages other than C.
FILENAME	CHAR(255)	File name to identify the load input source when using a shared library.

Table 5. Fields in the SQLU-LOCATION-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the *location_entry* field. For languages other than C.
LOCATION_ENTRY	CHAR(256)	Name of input data files for the load utility.

Valid values for *MEDIA_TYPE* (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices (tapes, disks, or diskettes)

SQLU_SERVER_LOCATION
Server devices (tapes, disks, or diskettes; load only). Can be specified only for the *pDataFileList* parameter.

SQLU_TSM_MEDIA
TSM

SQLU_OTHER_MEDIA
Vendor library

SQLU_USER_EXIT
User exit (OS/2 only)

SQLU_PIPE_MEDIA
Named pipe (for vendor APIs only)

SQLU_DISK_MEDIA
Disk (for vendor APIs only)

SQLU_DISKETTE_MEDIA
Diskette (for vendor APIs only)

SQLU_TAPE_MEDIA
Tape (for vendor APIs only).

Data Structure: SQLU-MEDIA-LIST

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_media_list
{
  char            media_type;
  char            filler[3];
  sqlint32            sessions;
  union sqlu_media_list_targets target;
} sqlu_media_list;
/* ... */

/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST-TARGETS */
/* ... */
union sqlu_media_list_targets
{
  struct sqlu_media_entry        *media;
  struct sqlu_vendor             *vendor;
  struct sqlu_location_entry     *location;
};
/* ... */

/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_media_entry
{
  sqluint32       reserve_len;
  char            media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;
/* ... */

/* File: sqlutil.h */
/* Structure: SQLU-VENDOR */
/* ... */
typedef SQL_STRUCTURE sqlu_vendor
{
  sqluint32       reserve_len1;
  char            shr_lib[SQLU_SHR_LIB_LEN+1];
  sqluint32       reserve_len2;
  char            filename[SQLU_SHR_LIB_LEN+1];
} sqlu_vendor;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-LOCATION-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_location_entry
{
  sqluint32       reserve_len;
  char            location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_entry;
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLU-MEDIA-LIST.
    05 SQL-MEDIA-TYPE         PIC X.
    05 SQL-FILLER             PIC X(3).
    05 SQL-SESSIONS           PIC S9(9) COMP-5.
    05 SQL-TARGET.
        10 SQL-MEDIA          USAGE IS POINTER.
        10 SQL-VENDOR         REDEFINES SQL-MEDIA
        10 SQL-LOCATION       REDEFINES SQL-MEDIA
        10 FILLER             REDEFINES SQL-MEDIA
*

* File: sqlutil.cbl
01 SQLU-MEDIA-ENTRY.
    05 SQL-MEDENT-LEN         PIC 9(9) COMP-5.
    05 SQL-MEDIA-ENTRY        PIC X(215).
    05 FILLER                 PIC X.
*

* File: sqlutil.cbl
01 SQLU-VENDOR.
    05 SQL-SHRLIB-LEN         PIC 9(9) COMP-5.
    05 SQL-SHR-LIB            PIC X(255).
    05 FILLER                 PIC X.
    05 SQL-FILENAME-LEN       PIC 9(9) COMP-5.
    05 SQL-FILENAME           PIC X(255).
    05 FILLER                 PIC X.
*

* File: sqlutil.cbl
01 SQLU-LOCATION-ENTRY.
    05 SQL-LOCATION-LEN       PIC 9(9) COMP-5.
    05 SQL-LOCATION-ENTRY     PIC X(255).
    05 FILLER                 PIC X.
*
```

Data Structure: SQLU-TABLESPACE-BKRST-LIST

This structure is used to provide a list of table space names.

Table 6. Fields in the SQLU-TABLESPACE-BKRST-LIST Structure

Field Name	Data Type	Description
NUM_ENTRY	INTEGER	Number of entries in the list pointed to by the *tablespace* field.
TABLESPACE	Pointer	A pointer to an *sqlu_tablespace_entry* structure.

Table 7. Fields in the SQLU-TABLESPACE-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the character string provided in the *tablespace_entry* field. For languages other than C.
TABLESPACE_ENTRY	CHAR(19)	Table space name.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLU-TABLESPACE-BKRST-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long            num_entry;
  struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;
/* ... */

/* File: sqlutil.h */
/* Structure: SQLU-TABLESPACE-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_tablespace_entry
{
  sqluint32       reserve_len;
  char            tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
  char            filler[1];
} sqlu_tablespace_entry;
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLU-TABLESPACE-BKRST-LIST.
    05 SQL-NUM-ENTRY          PIC S9(9) COMP-5.
    05 SQL-TABLESPACE         USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQLU-TABLESPACE-ENTRY.
    05 SQL-TBSP-LEN           PIC 9(9) COMP-5.
    05 SQL-TABLESPACE-ENTRY   PIC X(18).
    05 FILLER                 PIC X.
    05 SQL-FILLER             PIC X(1).
*
```

Example Backup Sessions

CLP Examples

```
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
```

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
```

A sample DB2 command script, and information on how to use it, are provided in "Appendix F. Recovery CLP Script" on page 425.

API Examples

Sample programs containing DB2 APIs and embedded SQL calls, and information on how to use them, are provided in "Appendix E. Recovery Sample Programs" on page 359.

Optimizing Backup Performance

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

- Specify table space backup.

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

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

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

- Increase the number of buffers.

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

Backup 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 for 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.

Troubleshooting Backup

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

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

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

The backup utility provides concurrency control for multiple processes that are making backup copies of different databases. This concurrency control keeps the backup target devices open until all the backup operations have ended. If an error occurs during a backup operation, and an open container

cannot be closed, other backup operations targeting the same drive may receive access errors. To correct such access errors, you must terminate the backup operation that caused the error and disconnect from the target device. If you are using the backup utility for concurrent backup operations to tape, ensure that the processes do not target the same tape.

Chapter 3. Database Restore

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

The following topics are covered:

Restore Overview

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

```
db2 restore db sample
```

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

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

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

A database restore operation requires an exclusive connection: that is, no applications can be running against the database when the operation starts,

and the restore utility prevents other applications from accessing the database until the restore operation completes successfully. A table space restore operation, however, can be done online.

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

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

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

Privileges, Authorities, and Authorization Required to Use Restore

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

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

Using Restore

Before Using Restore

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

The database can be local or remote.

Invoking Restore

The restore utility can be invoked through:

- The command line processor (CLP).

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

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

- The Restore Database notebook or wizard in the Control Center. To open the Restore Database notebook or wizard:

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

  For general information about the Control Center, see the *Administration Guide*. Detailed information is provided through the online help facility within the Control Center.

- An application programming interface (API), **sqlrestore**. For information about this API, see "Restore Database API" on page 116. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*.

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

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

You can redefine table space containers by invoking the RESTORE DATABASE command and specifying the REDIRECT parameter, or by using the Containers page of the Restore Database notebook in the Control Center. For redirected restore examples that use the command line processor, a command script, or the application programming interface, see "Example Restore Sessions" on page 126.

Redefining Table Space Containers (Redirected Restore)

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. Similarly, you could move a DMS table space from file containers to device containers.

Restoring to an Existing Database

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

A database *seed* is a unique identifier for a database that does not change during the life of the database. The seed is assigned by the database manager when the database is created. It remains unchanged following a restore operation, even if the backup image has a different database seed. 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. If the recovery history file is damaged, the database manager copies the file from the backup image.
- Retains the authentication type for the existing database.
- Retains the database directories for the existing database. The directories define where the database resides, and how it is cataloged.
- Compares the database seeds. If the seeds are different:
  - Deletes the logs associated with the existing database.
  - Copies the database configuration file from the backup image.
  - Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command.

  If the database seeds are the same:
  - Deletes the logs if the image is for a non-recoverable database.
  - Retains the current database configuration file, unless the file has been corrupted, in which case the file is copied from the backup image.

– Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command; otherwise, copies the current log path to the database configuration file. Validates the log path: If the path cannot be used by the database, changes the database configuration to use the default log path.

Restoring to a New Database

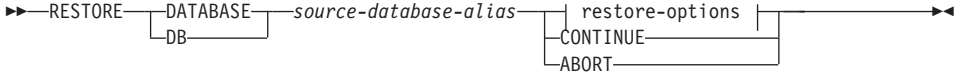
You can create a new database and then restore a full database backup image to it. The code pages of the backup image and the target database must match.

When restoring to a new database, the restore utility:

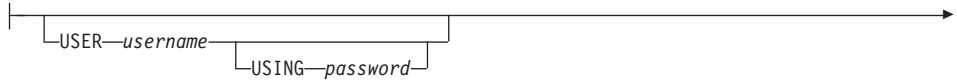
- Creates a new database, using the database alias name that was specified through the target database alias parameter. (If a target database alias was not specified, the restore utility creates the database with an alias that is the same as that specified through the source database alias parameter.)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

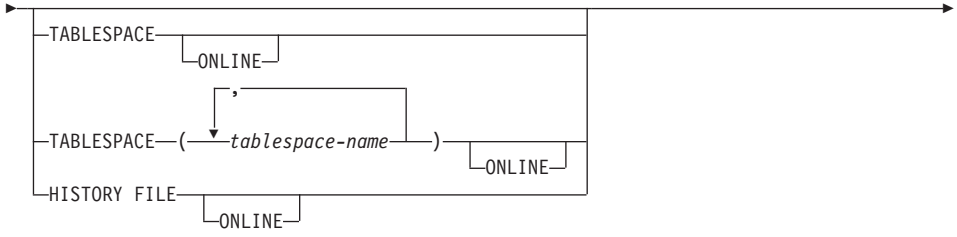
RESTORE DATABASE Command

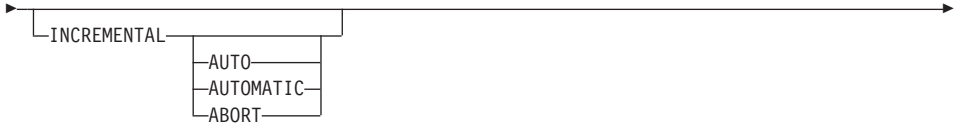
Command Syntax

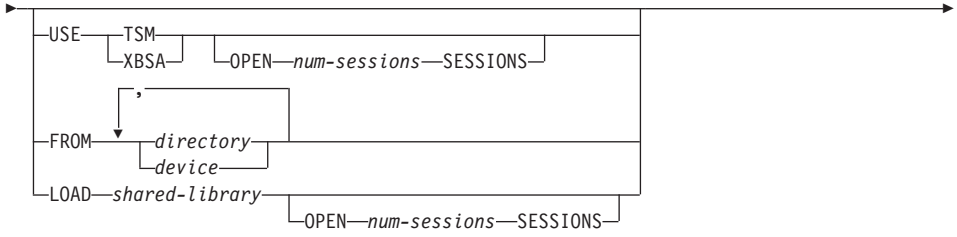


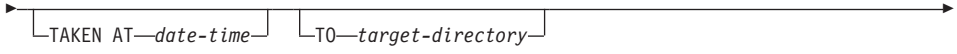
restore-options:

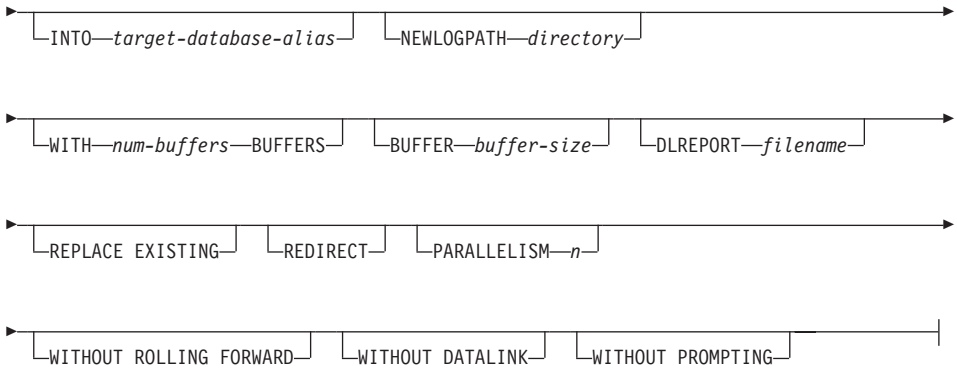












Command Parameters

DATABASE source-database-alias
Alias of the source database from which the backup was taken.

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

ABORT
This parameter:

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

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

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

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

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

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

INCREMENTAL
Specifies a manual cumulative restore operation. The user must issue each restore command manually.

AUTOMATIC/AUTO
Specifies an automatic cumulative (incremental) restore operation.

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

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

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

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

On Windows operating systems or OS/2, the specified directory must not be a DB2-generated directory. For example, given the following commands:

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

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

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

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

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

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

Tape is not supported on OS/2. On OS/2, `0` or `0:` can be specified to cause the restore utility to call a user exit program. (This can happen only if a user exit program was used to back up the database.) When restoring through a user exit program, the path to the database is the only reference used to locate the containers; therefore, all the containers for that database are restored.

Redirected restore is not allowed when a user exit program is being used.

LOAD shared-library
The name of the shared library (DLL on Windows operating systems or OS/2) containing the vendor backup and restore I/O functions to be used. The name can contain a full path. If the full path is not given, the value defaults to the path on which the user exit program resides.

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

TO target-directory
The target database directory. This parameter is ignored if the utility is restoring to an existing database.

Note: On Windows operating systems or OS/2, specify only the drive letter when using this parameter. If a longer path is specified, an error is returned.

INTO target-database-alias
The target database alias. If the target database does not exist, it will be created.

NEWLOGPATH directory
The fully qualified name of a directory that will be used for active log

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

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

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

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

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

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

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

REDIRECT
Specifies a redirected restore operation. To complete a redirected restore operation, this command should be followed by one or more

SET TABLESPACE CONTAINERS commands, and then by a RESTORE DATABASE command with the CONTINUE option.

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

WITHOUT ROLLING FORWARD
Specifies that the database is not to be put in rollforward pending state after it has been successfully restored.

If, following a successful restore operation, the database is in rollforward pending state, the "ROLLFORWARD DATABASE Command" on page 142 must be invoked before the database can be used again.

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

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

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

Restore Database API

C API Syntax

```
/* File: sqlutil.h */
/* API: Restore Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurestore (
    char * pSourceDbAlias,
    char * pTargetDbAlias,
    sqluint32 BufferSize,
    sqluint32 RollforwardMode,
    sqluint32 DatalinkMode,
    sqluint32 RestoreType,
    sqluint32 RestoreMode,
    sqluint32 CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    char * pTargetPath,
    sqluint32 NumBuffers,
    char * pReportFile,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaSourceList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    sqluint32 VendorOptionsSize,
    void * pVendorOptions,
    sqluint32 Parallelism,
    void * pRestoreInfo,
    void * pContainerPageList,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Restore Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrestore (
   unsigned short SourceDbAliasLen,
   unsigned short TargetDbAliasLen,
   unsigned short TimestampLen,
   unsigned short TargetPathLen,
   unsigned short UserNameLen,
   unsigned short PasswordLen,
   unsigned short ReportFileLen,
   unsigned short Reserved2Len,
   char * pSourceDbAlias,
   char * pTargetDbAlias,
   sqluint32 BufferSize,
   sqluint32 RollforwardMode,
   sqluint32 DatalinkMode,
   sqluint32 RestoreType,
   sqluint32 RestoreMode,
   sqluint32 CallerAction,
   char * pApplicationId,
   char * pTimestamp,
   char * pTargetPath,
   sqluint32 NumBuffers,
   char * pReportFile,
   struct sqlu_tablespace_bkrst_list * pTablespaceList,
   struct sqlu_media_list * pMediaSourceList,
   char * pUserName,
   char * pPassword,
   void * pReserved2,
   sqluint32 VendorOptionsSize,
   void * pVendorOptions,
   sqluint32 Parallelism,
   unsigned short RestoreInfoSize,
   void * pRestoreInfo,
   unsigned short ContainerPageListSize,
   void * pContainerPageList,
   void * pReserved3,
   struct sqlca * pSqlca);
/* ... */
```

API Parameters

SourceDbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the source database alias.

TargetDbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the target database alias. Set to zero if no target database alias is specified.

TimestampLen
Input. A 2-byte unsigned integer representing the length in bytes of the time stamp. Set to zero if no time stamp is provided.

TargetPathLen
Input. A 2-byte unsigned integer representing the length in bytes of the target directory. Set to zero if no target path is provided.

UserNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

ReportFileLen
Input. A 2-byte unsigned integer representing the length in bytes of the report file name. Set to zero if no report file name is provided.

Reserved2Len
Input. A 2-byte unsigned integer representing the length in bytes of the reserved area. Set to zero.

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

pTargetDbAlias
Input. A string containing the target database alias. If the value of this parameter is NULL, the value of *pSourceDbAlias* is used.

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

The specified buffer size must be equal to or an integer multiple of the buffer size used to produce the backup image.

RollforwardMode
Input. Specifies whether or not to put the database in rollforward pending state at the end of the restore operation. Valid values (defined in `sqlutil`) are:

SQLUD_ROLLFWD
Put the database in rollforward pending state after it has been successfully restored.

SQLUD_NOROLLFWD
Do not put the database 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 Database API" on page 148 must be invoked before the database can be used again.

DatalinkMode
Input. Specifies whether any tables with DATALINK columns are to be put in DataLink_Reconcile_Pending (DRP) state, and whether reconciliation of linked files is to be performed. Valid values (defined in `sqlutil`) are:

SQLUD_DATALINK
Perform reconciliation operations. Tables with a defined DATALINK column must have the RECOVERY YES option specified.

SQLUD_NODATALINK
Do not perform reconciliation operations. Tables with DATALINK columns are put in DRP state. Tables with a defined DATALINK column must have the RECOVERY YES option specified.

RestoreType
Input. Specifies the type of restore operation. Valid values (defined in `sqlutil`) are:

SQLUD_FULL
Restore everything from the backup image. This is run offline. This value will not be supported in the future. Use `SQLUD_DB` to specify a full database restore operation.

SQLUD_DB
Restore all table spaces in the database. This is run offline.

SQLUD_ONLINE_TABLESPACE
Restore only table space-level backup images. This is run online. This value will not be supported in the future. Use `SQLUD_TABLESPACE | SQLUD_ONLINE` to specify an online table space-level restore operation.

SQLUD_TABLESPACE
Restore only the table space-level backup images. This can be run online or offline.

SQLUD_HISTORY
Restore only the recovery history file.

SQLUD_INCREMENTAL
Perform a manual cumulative restore operation.

SQLUD_AUTOMATIC
Perform an automatic cumulative (incremental) restore operation. Must be specified with SQLUD_INCREMENTAL; that is, SQLUD_INCREMENTAL | SQLUD_AUTOMATIC.

RestoreMode
Input. Specifies whether the restore operation is to be performed offline or online. Valid values (defined in sqlutil) are:

SQLUD_OFFLINE
Perform an offline restore operation.

SQLUD_ONLINE
Perform an online restore operation.

CallerAction
Input. Specifies the type of action to be taken. Valid values (defined in sqlutil) are:

SQLUD_RESTORE
Start the restore operation.

SQLUD_TERMINATE_INCREMENTAL
Terminate an incremental restore operation before completion.

SQLUD_NOINTERRUPT
Start the restore operation. Specifies that the restore operation is to run unattended. 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, when all of the media required for the restore operation are known to have been mounted, and utility prompts are not desired.

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

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

SQLUD_DEVICE_TERMINATE
Remove a device from the list of devices being used by the restore utility. When a device has exhausted its input, the restore utility returns a warning to the caller. Invoke the

restore utility again with this caller action. The device that generated the warning is removed from the list of devices being used.

SQLUD_PARM_CHECK
Validate parameters without performing the restore operation.

SQLUD_RESTORE_STORDEF
Initial call. Request table space container redefinition.

CallerAction must be set to `SQLUD_RESTORE`, `SQLUD_NOINTERRUPT`, `SQLUD_RESTORE_STORDEF`, or `SQLUD_PARM_CHECK` on the first call.

pApplicationId
Output. Supply a buffer of length `SQLU_APPLID_LEN+1` (defined in `sqlutil`). The restore utility returns a string identifying the agent that is servicing the application. Can be used with the database system monitor APIs to monitor the application.

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

pTargetPath
Input. A string containing the relative or fully qualified name of the target database directory. Used if a new database is to be created during the restore operation.

NumBuffers
Input. The number of buffers to be used for the restore operation.

pReportFile
The file name, if specified, must be fully qualified. 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.

pTablespaceList
Specifies one or more table spaces to be restored. Used when restoring a subset of the backup image, or a table space from a table space-level backup image.

The following restrictions apply:

- The database must be recoverable. Recoverable databases have either the *logretain* database configuration parameter set to "RECOVERY", the *userexit* database configuration parameter enabled, or both.

- The database being restored to must be the same database that was used to create the backup image. That is, table spaces cannot be added to a database through the table space restore function.
- This function is not available when restoring from a user exit on OS/2.
- The rollforward utility ensures that table spaces restored in an MPP environment are synchronized with any other node containing the same table spaces.

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

pMediaSourceList
Input. Source media for the backup image. See "Data Structure: SQLU-MEDIA-LIST" on page 96. The information that the caller needs to provide in this structure is dependent upon the value of the *media_type* field. Valid values for this field (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks, or diskettes). Provide a list of *sqlu_media_entry* structures. On Windows operating systems or OS/2, the entries can only be directory paths, not tape device names.

SQLU_TSM_MEDIA
TSM. No additional input is required. The TSM shared library provided with DB2 is used. If a different version of TSM is desired, use `SQLU_OTHER_MEDIA`, and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in an *sqlu_vendor* structure.

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

pUserName
Input. A string containing the user name to be used for a connection.

pPassword
Input. A string containing the password used to authenticate the user name.

pReserved2
Reserved for future use.

VendorOptionsSize
Input. The length of the vendor options field. The length cannot exceed 65535 bytes.

pVendorOptions
Input. To be used by the vendor 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 for this data is not checked.

Parallelism
Input. Degree of intra-partition parallelism (number of buffer manipulators).

RestoreInfoSize
Reserved for future use.

pRestoreInfo
Reserved for future use.

ContainerPageListSize
Reserved for future use.

pContainerPageList
Reserved for future use.

pReserved3
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

REXX API Syntax

```
RESTORE DATABASE source-database-alias [USING :value] [USER username USING password]
[TABLESPACE :tablespacenames] [ONLINE | HISTORY FILE ]
[LOAD shared-library [OPTIONS vendor-options] [OPEN num-sessions SESSIONS] |
 FROM :source-area | USE TSM [OPEN num-sessions SESSIONS] | USER_EXIT]
[TAKEN AT timestamp] [TO target-directory] [INTO target-database-alias]
[ACTION caller-action] [WITH num-buffers BUFFERS] [BUFFERSIZE buffer-size]
[WITHOUT ROLLING FORWARD] [PARALLELISM parallelism-degree]
```

REXX API Parameters

source-database-alias
Alias of the source database from which the database backup image was taken.

value A compound REXX host variable to which the database restore information is returned. In the following, XXX represents the host variable name:

XXX.0 Number of elements in the variable (always 1)

XXX.1 An application ID that identifies the agent that serves the application.

username
Identifies the user name to be used for a connection.

password
The password used to authenticate the user name.

tablespacenames
A compound REXX host variable containing a list of table spaces to be restored. In the following, XXX is the name of the host variable:

XXX.0 Number of table spaces to be restored

XXX.1 First table space name

XXX.2 Second table space name

XXX.3 and so on.

HISTORY FILE
Specifies to restore only the history file from the backup image.

shared-library
The name of the shared library (DLL on Windows operating systems or OS/2) containing the vendor restore I/O functions to be used. It may contain the full path. If the full path is not given, defaults to the path on which the user exit program resides.

vendor-options
Information required by the vendor functions.

num-sessions
The number of I/O sessions to be used with TSM or the vendor product.

source-area
A compound REXX host variable that indicates on which directory or device the backup image resides. The default value is the current directory. On Windows operating systems or OS/2, the entries can only be directory paths, not tape device names.

timestamp
The time stamp of the database backup image.

target-directory
The target database directory.

target-database-alias
The target database alias. If the target database does not exist, it will be created.

caller-action
Specifies action to be taken. Valid values are:

SQLUD_RESTORE
Start the restore operation.

SQLUD_NOINTERRUPT
Start the restore operation. Specifies that the restore operation is to run unattended. 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, when all of the media required for the restore operation are known to have been mounted, and utility prompts are not desired.

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

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

SQLUD_DEVICE_TERMINATE
Remove a device from the list of devices being used by the restore utility. When a device has exhausted its input, the restore utility returns a warning to the caller. Invoke the restore utility again with this caller action. The device that generated the warning is removed from the list of devices being used.

SQLUD_PARM_CHECK
Validate parameters without performing the restore operation.

SQLUD_RESTORE_STORDEF
Initial call. Request table space container redefinition.

num-buffers
Number of backup buffers to be used.

buffer-size
Backup buffer size in 4-KB allocation units. Minimum value is 16 units.

parallelism-degree
Degree of intra-partition parallelism (number of buffer manipulators).

Example Restore Sessions

CLP Examples

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

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

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

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

   ```
   db2 restore db mydb abort
   ```

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

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

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

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

   ```
   db2 restore db mydb continue
   ```

   This is the final step of the redirected restore operation.

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

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

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

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

```
restore db kdr incremental automatic taken at (Thu)
```

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

```
restore db kdr incremental taken at (Thu)
restore db kdr incremental taken at (Sun)
restore db kdr incremental taken at (Wed)
restore db kdr incremental taken at (Thu)
```

A sample DB2 command script, and information on how to use it, are provided in "Appendix F. Recovery CLP Script" on page 425.

API Examples

Sample programs containing DB2 APIs and embedded SQL calls, and information on how to use them, are provided in "Appendix E. Recovery Sample Programs" on page 359.

Optimizing Restore Performance

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

- Increase the restore buffer size.

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

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

Restore 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.
- If you are using the DB2 Control Center, you cannot restore backup images that were created with previous versions of DB2.
- A database restore operation cannot be started while the rollforward process is running.
- You can restore a table space only if the table space currently exists, and if it is the same table space; "same" means that the table space was not dropped and then recreated between the backup and the restore operation.)
- You cannot restore a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.
- On OS/2, a partial or subset restore operation is not possible when calling a user exit program.
- If the code page of the database being restored does not match a code page that is available to the application, or if the database manager does not

support code page conversion from the database code page to a code page that is available to the application, the restored database will not be usable.

Troubleshooting Restore

If SQL0970N is returned when you try to restore a backup image to a new database on a different system, and your original database has table spaces defined using an absolute path, use a redirected restore operation to define table space containers on the new system. The restore utility attempts to use the absolute path and containers that do not exist on your new system.

If a system failure occurs during a database restore operation, you cannot connect to the database until you invoke the restore utility again, and successfully complete the restore operation. If a system failure occurs during a table space restore operation, only the table space being restored is unusable. The other table spaces in the database can still be used.

Chapter 4. Rollforward Recovery

This section describes the DB2 UDB rollforward utility, which is used to recover a database by applying transactions that were recorded in the database recovery log files.

The following topics are covered:

Rollforward Overview

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 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.
```

For an explanation of these fields, see "ROLLFORWARD DATABASE Command" on page 142.

The general approach to rollforward recovery involves:

1. Invoking the rollforward utility without the STOP option
2. Invoking the rollforward utility with the QUERY STATUS option

   If you specify recovery to the end of the logs, the QUERY STATUS option can indicate that one or more log files is missing, if the returned point in time is earlier than you expect.

   If you specify point-in-time recovery, the QUERY STATUS option will help you to ensure that the rollforward operation has completed at the correct point.
3. Invoking the rollforward utility with the STOP option. After the operation stops, it is not possible to roll additional changes forward.

A database must be restored successfully (using the restore utility) before it can be rolled forward, but a table space does not. A table space may be temporarily put in rollforward pending state, but not require a restore operation to undo it (following a power interruption, for example).

When the rollforward utility is invoked:

- If the database is in rollforward pending state, the database is rolled forward. If table spaces are also in rollforward pending state, you must invoke the rollforward utility again after the database rollforward operation completes to roll the table spaces forward.
- If the database is *not* in rollforward pending state, but table spaces in the database *are* in rollforward pending state:
  - If you specify a list of table spaces, only those table spaces are rolled forward.
  - If you do not specify a list of table spaces, all table spaces that are in rollforward pending state are rolled forward.

A database rollforward operation runs offline. The database is not available for use until the rollforward operation completes successfully, and the operation cannot complete unless the STOP option was specified when the utility was invoked.

A table space rollforward operation can run offline. The database is not available for use until the rollforward operation completes successfully. This occurs if the end of the logs is reached, or if the STOP option was specified when the utility was invoked.

You can perform an *online* rollforward operation on table spaces, as long as SYSCATSPACE is not included. When you perform an online rollforward operation on a table space, the table space is not available for use, but the other table spaces in the database *are* available.

When you first create a database, it is enabled for circular logging only. This means that logs are reused, rather than being saved or archived. With circular logging, rollforward recovery is not possible: only crash recovery or version recovery can be done (see "Understanding Recovery Logs" on page 30). Archived logs document changes to a database that occur after a backup was taken. You enable log archiving (and rollforward recovery) by setting the *logretain* database configuration parameter to `RECOVERY`, or setting the *userexit* database configuration parameter to `YES`, or both. The default value for both of these parameters is `NO`, because initially, there is no backup image that you can use to recover the database. When you change the value of one or both of these parameters, the database is put into backup pending state, and you must take an offline backup of the database before it can be used again.

For more information about the database configuration parameters associated with logging, see "Configuration Parameters for Database Logging" on page 37.

Privileges, Authorities, and Authorization Required to Use Rollforward

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

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

Using Rollforward

Before Using Rollforward

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.

The database can be local or remote.

Invoking Rollforward

The rollforward utility can be invoked through:

- The command line processor (CLP).

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

  ```
  db2 rollforward db sample to end of logs and stop
  ```

- The Rollforward Database notebook in the Control Center. To open the Rollforward Database notebook:

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

  For general information about the Control Center, see the *Administration Guide*. Detailed information is provided through the online help facility within the Control Center.

- An application programming interface (API), **sqluroll**. For information about this API, see "Rollforward Database API" on page 148. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*.

In a partitioned database environment, the rollforward utility must be invoked from the catalog node of the database.

Rolling Forward Changes in a Table Space

If the database is enabled for forward recovery, you have the option of backing up, restoring, and rolling forward table spaces instead of the entire database. You may want to implement a recovery strategy for individual table spaces because this can save time: it takes less time to recover a portion of the database than it does to recover the entire database. For example, if a disk is

bad, and it contains only one table space, that table space can be restored and rolled forward without having to recover the entire database, and without impacting user access to the rest of the database, unless the damaged table space contains the system catalog tables; in this situation, you cannot connect to the database. (The system catalog table space can be restored independently if a table space-level backup image containing the system catalog table space is available.) Table space-level backups also allow you to back up critical parts of the database more frequently than other parts, and requires less time than backing up the entire database.

After a table space is restored, it is always in rollforward pending state. To make the table space usable, you must perform rollforward recovery on it. In most cases, you have the option of rolling forward to the end of the logs, or rolling forward to a point in time. You cannot, however, roll table spaces containing system catalog tables forward to a point in time. These table spaces must be rolled forward to the end of the logs to ensure that all table spaces in the database remain consistent.

Before rolling a table space forward, invoke the LIST TABLESPACES SHOW DETAIL command. This command returns the *minimum recovery time*, which is the earliest point in time to which the table space can be rolled forward. The minimum recovery time is updated when data definition language (DDL) statements are run against the table space, or against tables in the table space. The table space must be rolled forward to at least the minimum recovery time, so that it becomes synchronized with the information in the system catalog tables. If recovering more than one table space, the table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces being recovered. In a partitioned database environment, issue the LIST TABLESPACES SHOW DETAIL command on all partitions. The table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces on all partitions.

If you are rolling table spaces forward to a point in time, and a table is contained in multiple table spaces, all of these table spaces must be rolled forward simultaneously. If, for example, the table data is contained in one table space, and the index for the table is contained in another table space, you must roll both table spaces forward simultaneously to the same point in time.

If the data and the long objects in a table are in separate table spaces, and the table 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 summary table that is in another table space
- A summary 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 summary table is placed in check pending state at the end of the rollforward operation.

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, both table spaces will be in check pending state at the end of the point-in-time rollforward operation. 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.)

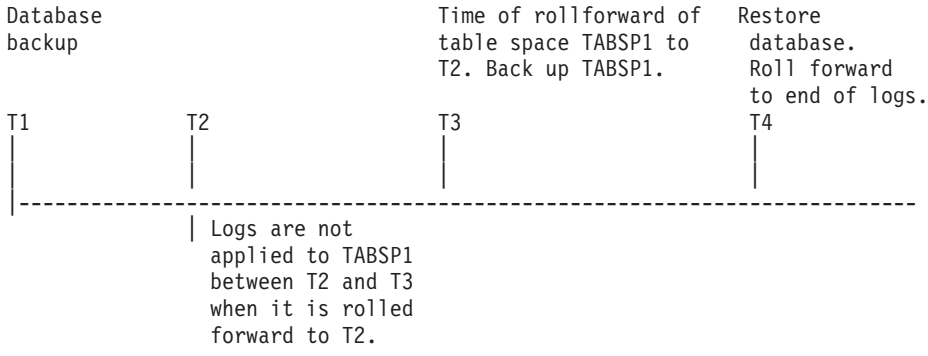


Figure 17. Table Space Backup Requirement

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.

Recovering a Dropped Table

You may occasionally drop a table whose data you still need. If this is the case, you should consider making your critical tables recoverable following a drop table operation.

You could recover the table data by invoking a database restore operation, followed by a database rollforward operation to a point in time before the table was dropped. This may be time consuming if the database is large, and your data will be unavailable during recovery.

DB2's dropped table recovery feature lets you recover your dropped table data using table space-level restore and rollforward operations. This will be faster than database-level recovery, and your database will remain available to users.

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 (see the *SQL Reference*). 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.

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.

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 (see "LIST HISTORY" on page 318). The dropped table ID is listed in the Backup ID column.
2. Restore a database- or table space-level backup image taken before the table was dropped.
3. Create an export directory to which files containing the table data are to be written. This directory must either be accessible to all database partitions, or exist on each partition. Subdirectories under this export directory are created automatically by each database partition. These subdirectories are named `NODE`*`nnnn`*, where *nnnn* represents the database partition or node number. Data files containing the dropped table data as it existed on each database partition are exported to a lower subdirectory called `data`. For example, `\export_directory\NODE0000\data`.
4. Roll forward to a point in time after the table was dropped, using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. Alternatively, roll forward to the end of the logs, so that updates to other tables in the table space or database are not lost.
5. Recreate the table using the CREATE TABLE statement from the recovery history file.
6. Import the table data that was exported during the rollforward operation into the table.

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 long 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 long table spaces.

- The metadata associated with row types. (The data is recovered, but not the metadata.) The data in the hierarchy table of the typed table will be recovered. This data may contain more information than appeared in the typed table that was dropped.

The names of linked files associated with DATALINK columns *can* be recovered. After importing the table data, the table should be reconciled with the DB2 Data Links Manager. Backup images of the files may or may not be restored by the DB2 Data Links Manager, depending on whether garbage collection has already deleted them.

Using the Load Copy Location File

The DB2LOADREC registry variable is used to identify the file with the load copy location information. This file is used during rollforward recovery to locate the load copy. It has information about:

- Media type
- Number of media devices to be used
- Location of the load copy generated during a table load operation
- File name of the load copy, if applicable

If the location file does not exist, or no matching entry is found in the file, the information from the log record is used.

The information in the file may be overwritten before rollforward recovery takes place.

Notes:

1. In a partitioned database environment, the DB2LOADREC registry variable must be in the `db2profile` file.
2. In a partitioned database environment, the load copy file must exist at each database partition server, and the file name (including the path) must be the same.
3. If an entry in the file identified by the DB2LOADREC registry variable is not valid, the old load copy location file is used to provide information to replace the invalid entry.

The following information is provided in the location file. The first five parameters must have valid values, and are used to identify the load copy. The entire structure is repeated for each load copy recorded. For example:

```
TIMestamp      19950725182542         * Time stamp generated at load time
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    TORONTO                * DB2INSTANCE
BUFfernumber   NULL                   * Number of buffers to be used for recovery
SESsionnumber  NULL                   * Number of sessions to be used for recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                        A for TSM
                                                        O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            O
SHRlib         /@sys/lib/backup_vendor.a
```

Notes:

1. The first three characters in each keyword are significant. All keywords are required in the specified order. Blank lines are not accepted.
2. The time stamp is in the form *yyyymmddhhmmss*.
3. All fields are mandatory, except for BUF and SES, which can be NULL. If SES is NULL, the value specified by the *numloadrecses* configuration parameter is used. If BUF is NULL, the default value is SES+2.
4. If even one of the entries in the location file is invalid, the previous load copy location file is used to provide those values.
5. The media type can be local device (`L` for tape, disk or diskettes), TSM (`A`), or other vendor (`O`). If the type is `L`, the number of locations, followed by the location entries, is required. If the type is `A`, no further input is required. If the type is `O`, the shared library name is required. For details about using TSM and other vendor products as backup media, see "Appendix G. Tivoli Storage Manager" on page 433.
6. The SHRlib parameter points to a library that has a function to store the load copy data.

7. If you invoke a load operation, specifying the COPY NO or the NONRECOVERABLE option, and do not take a backup copy of the database or affected table spaces after the operation completes, you cannot restore the database or table spaces to a point in time that follows the load operation. That is, you cannot use rollforward recovery to rebuild the database or table spaces to the state they were in following the load operation. You can only restore the database or table spaces to a point in time that precedes the load operation.

If you want to use a particular load copy, you can use the recovery history file for the database to determine the time stamp for that specific load operation. In a partitioned database environment, the recovery history file is local to each database partition.

For detailed information about the load utility, see the *Data Movement Utilities Guide and Reference*.

Synchronizing Clocks in a Partitioned Database System

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database system, DB2 uses the system clock on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 1999 when the year is 1997, and assume that the mistake is corrected *after* an update transaction is committed in the partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 1997 and November 7, 1999 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 1999, and the log clock remains at November 7, 1999 until the system clock matches this time. If you attempt to roll forward

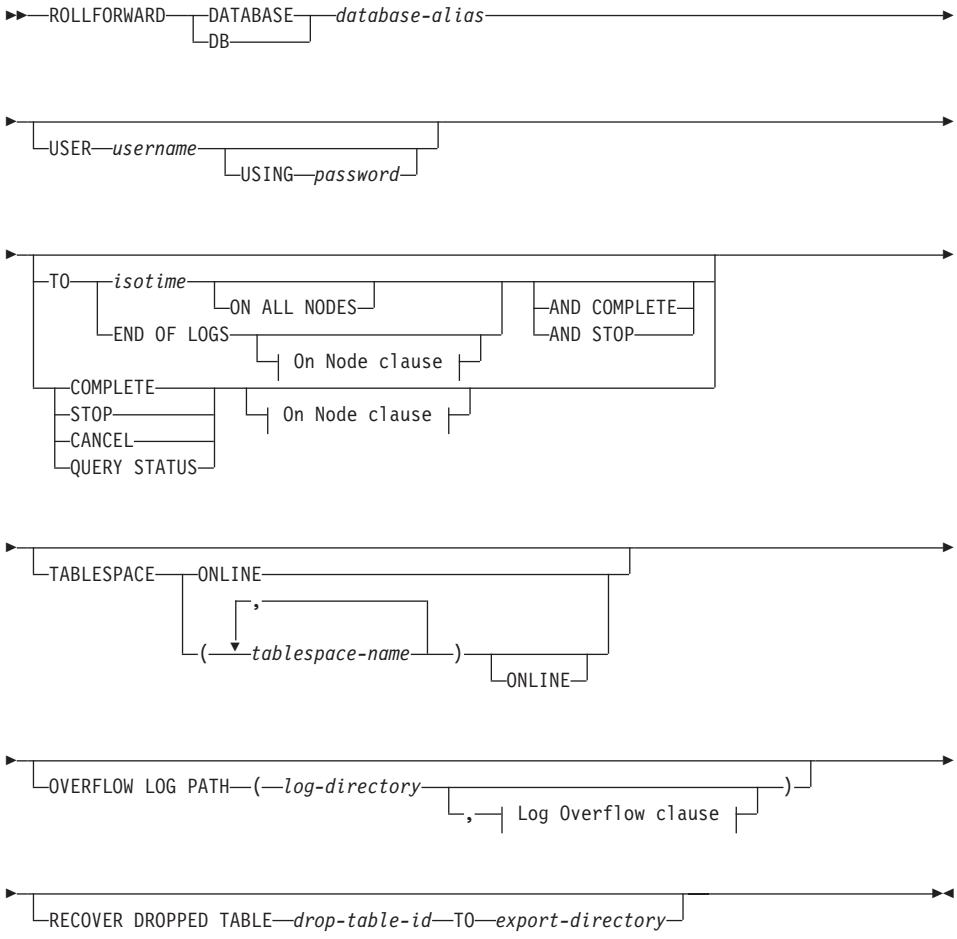
to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 1997.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

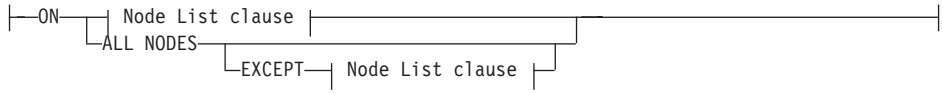
- The configurable values for this parameter range from 1 minute to 24 hours. For more information about setting *max_time_diff*, see the *Administration Guide: Performance* book.
- When the first connection request is made to a non-catalog node, the database partition server sends its time to the catalog node for the database. The catalog node then checks that the time on the node requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

ROLLFORWARD DATABASE Command

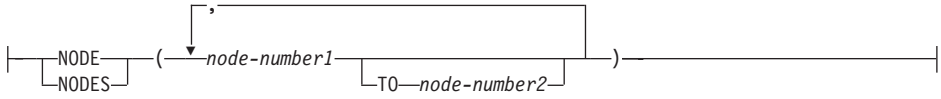
Command Syntax



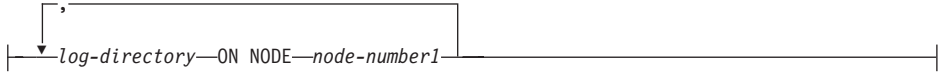
On Node clause:



Node 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, the user is prompted to enter it.

TO

isotime
The point in time to which all committed transactions are to be rolled forward (including the transaction committed precisely at that time, as well as all transactions committed previously).

This value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds), expressed in Coordinated Universal Time (UTC). UTC helps to avoid having the same time stamp associated with different logs (because of a change in time associated with daylight savings time, for example). The time stamp in a backup image is based on the local time at which the backup operation started. The CURRENT TIMEZONE special register specifies the difference between UTC and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits represent the number of hours, the next two digits represent the number of minutes, and the last two digits represent the number of seconds). Subtracting CURRENT TIMEZONE from a local time converts that local time to UTC.

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 NODES
Specifies that transactions are to be rolled forward on all nodes specified in the `db2nodes.cfg` file. This is the default if a node clause is not specified.

EXCEPT
Specifies that transactions are to be rolled forward on all nodes specified in the `db2nodes.cfg` file, except those specified in the node list.

ON NODE / ON NODES
Roll the database forward on a set of nodes.

node-number1
Specifies a node number in the node list.

node-number2
Specifies the second node number, so that all nodes from *node-number1* up to and including *node-number2* are included in the node list.

COMPLETE / STOP
Stops the rolling forward of log records, and completes the rollforward recovery process by rolling back any incomplete transactions and turning off the rollforward pending state of the database. This allows access to the database or table spaces that are being rolled forward. These keywords are equivalent; specify one or the other, but not both. The keyword AND permits specification of multiple operations at once; for example, `db2 rollforward db sample to end of logs and complete`.

Note: When rolling table spaces forward to a point in time, the table spaces are placed in backup pending state.

CANCEL
Cancels the rollforward recovery operation. This puts the database or one or more table spaces on all nodes on which forward recovery has been started in restore pending state:

- If a *database* rollforward operation is not in progress (that is, the database is in rollforward pending state), this option puts the database in restore pending state.
- If a *table space* rollforward operation is not in progress (that is, the table spaces are in rollforward pending state), a table space list must be specified. All table spaces in the list are put in restore pending state.

- If a table space rollforward operation *is* in progress (that is, at least one table space is in rollforward in progress state), all table spaces that are in rollforward in progress state are put in restore pending state. If a table space list is specified, it must include all table spaces that are in rollforward in progress state. All table spaces on the list are put in restore pending state.
- If rolling forward to a point in time, any table space name that is passed in is ignored, and all table spaces that are in rollforward in progress state are put in restore pending state.
- If rolling forward to the end of the logs with a table space list, only the table spaces listed are put in restore pending state.

This option cannot be used to cancel a rollforward operation *that is actually running*. It can only be used to cancel a rollforward operation that is in progress but not actually running at the time. A rollforward operation can be in progress but not running if:

- It terminated abnormally.
- The STOP option was not specified.
- An error caused it to fail. Some errors, such as rolling forward through a non-recoverable load operation, can put a table space into restore pending state.

Note: Use this option with caution, and only if the rollforward operation that is in progress cannot be completed because some of the table spaces have been put in rollforward pending state or in restore pending state. When in doubt, use the LIST TABLESPACES command to identify the table spaces that are in rollforward in progress state, or in rollforward pending state.

QUERY STATUS

Lists the log files that the database manager has rolled forward, the next archive file required, and the time stamp (in CUT) of the last committed transaction since rollforward processing began. In a partitioned database environment, this status information is returned for each node. The information returned contains the following fields:

Node number

Rollforward status

Status can be: database or table space rollforward pending, database or table space rollforward in progress, database or table space rollforward processing STOP, or not pending.

Next log file to be read

A string containing the name of the next required log file. In a partitioned database environment, use this information if the

rollforward utility fails with a return code indicating that a log file is missing or that a log information mismatch has occurred.

Log files processed
A string containing the names of processed log files that are no longer needed for recovery, and that can be removed from the directory. If, for example, the oldest uncommitted transaction starts in log file *x*, the range of obsolete log files will not include *x*; the range ends at *x* - 1.

Last committed transaction
A string containing a time stamp in ISO format (*yyyy-mm-dd-hh.mm.ss*). This time stamp marks the last transaction committed after the completion of rollforward recovery. The time stamp applies to the database. For table space rollforward recovery, it is the time stamp of the last transaction committed to the database.

Note: QUERY STATUS is the default value if the TO, STOP, COMPLETE, or CANCEL clauses are omitted. If TO, STOP, or COMPLETE was specified, status information is displayed if the command has completed successfully. If individual table spaces are specified, they are ignored; the status request does not apply only to specified table spaces.

TABLESPACE
This keyword is specified for table space-level rollforward recovery.

tablespace-name
Mandatory for table space-level rollforward recovery to a point in time. Allows a subset of table spaces to be specified for rollforward recovery to the end of the logs. In a partitioned database environment, each table space in the list does not have to exist at each node that is rolling forward. If it *does* exist, it must be in the correct state.

ONLINE
This keyword is specified to allow table space-level rollforward recovery to be done online. This means that other agents are allowed to connect while rollforward recovery is in progress.

OVERFLOW LOG PATH log-directory
Specifies an alternate log path to be searched for archived logs during recovery. Use this parameter if log files were moved to a location other than that specified by the *logpath* database configuration parameter. In a partitioned database environment, this is the (fully qualified) default overflow log path *for all nodes*. A relative overflow log path can be specified for single-partition databases. 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.

log-directory ON NODE
In a partitioned database environment, allows a different log path to override the default overflow log path for a specific node.

RECOVER DROPPED TABLE drop-table-id
Recovers a dropped table during the rollforward operation. The table ID can be obtained using "LIST HISTORY" on page 318.

TO export-directory
Specifies a directory to which files containing the table data are to be written. The directory must be accessible to all nodes.

Rollforward Database API

C API Syntax

```
/* File: sqlutil.h */
/* API: Rollforward Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqluroll (
    struct rfwd_input * pRfwdInput,
    struct rfwd_output * pRfwdOuput,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Rollforward Database */
/* ... */
SQL_API_RC SQL_API_RN
  sqlgroll (
    struct grfwd_input * grfwdin,
    struct rfwd_output * rfwdout,
    struct sqlca * sqlca);

SQL_STRUCTURE grfwd_input
{
   unsigned short DbAliasLen,
   unsigned short StopTimeLen,
   unsigned short UserNameLen,
   unsigned short PasswordLen,
   unsigned short OverflowLogPathLen,
   unsigned short ReportFileLen,      /* NOTE: This parameter is no longer used */
                                      /*       for the DB2 Data Links Manager. */
   sqluint32 Version,
   char * pDbAlias,
   unsigned short CallerAction,
   char * pStopTime,
   char * pUserName,
   char * pPassword,
   char * pOverflowLogPath,
   unsigned short NumChngLgOvrflw,
   struct sqlurf_newlogpath * pChngLogOvrflw,
   unsigned short ConnectMode,
   struct sqlu_tablespace_bkrst_list * pTablespaceList,
   short AllNodeFlag,
   short NumNodes,
   SQL_PDB_NODE_TYPE * pNodeList,
   short NumNodeInfo,
   unsigned short DlMode,             /* NOTE: This parameter is no longer used */
                                      /*       for the DB2 Data Links Manager. */
   char * pReportFile,                /* NOTE: This parameter is no longer used */
                                      /*       for the DB2 Data Links Manager. */
   char * pDroppedTblID,
   char * pExportDir
}
/* ... */
```

API Parameters

pRfwdInput
Input. A pointer to the *rfwd_input* structure. For more information about this structure, see "Data Structure: RFWD-INPUT" on page 157.

pRfwdOutput
Output. A pointer to the *rfwd_output* structure. For more information about this structure, see "Data Structure: RFWD-OUTPUT" on page 160.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

StopTimeLen
Input. A 2-byte unsigned integer representing the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

UserNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

OverflowLogPathLen
Input. A 2-byte unsigned integer representing the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

ReportFileLen
Input. This parameter is not currently used, and should be set to zero.

Version
Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

pDbAlias
Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

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

SQLUM_ROLLFWD
Roll forward to the point in time specified by *pPointInTime*. For database rollforward recovery, the database is left in rollforward pending state. For table space-level rollforward to a point in time, the table spaces are left in rollforward-in-progress state.

SQLUM_STOP
End rollforward recovery. No new log records are processed and uncommitted transactions are backed out. The

rollforward-pending state of the database or table spaces is turned off. Synonym is SQLUM_COMPLETE.

SQLUM_ROLLFWD_STOP
Roll forward to the point in time specified by *pPointInTime*, and end rollforward recovery. The rollforward pending state of the database or table spaces is turned off. Synonym is SQLUM_ROLLFWD_COMPLETE.

SQLUM_QUERY
Query values for *pNextArcFileName*, *pFirstDelArcFileName*, *pLastDelArcFileName*, and *pLastCommitTime*. Return database status and a node number.

SQLUM_PARM_CHECK
Validate parameters without performing the rollforward operation.

SQLUM_CANCEL
Cancel the rollforward operation that is currently running. The database or table space are put in recovery pending state.

Note: This option cannot be used while the rollforward operation is actually running. It can be used if the operation is paused (that is, waiting for a STOP), or if a system failure occurred during the rollforward operation. It should be used with caution.

Rolling databases forward may require a load recovery operation using tape devices. The rollforward API returns with a warning message if user intervention on a device is required. The API can be called again with one of the following three caller actions:

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

SQLUM_LOADREC_DEVICE_TERMINATE
Stop using the device that generated the warning message (for example, when there are no more tapes).

SQLUM_LOADREC_TERMINATE
Terminate all devices being used by load recovery.

pStopTime
Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for `SQLUM_QUERY`, `SQLUM_PARM_CHECK`, and any of the load recovery (`SQLUM_LOADREC_xxx`) caller actions.

pUserName
Input. A string containing the user name of the application. May be NULL.

pPassword
Input. A string containing the password of the supplied user name (if any). May be NULL.

pOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the *logpath* (see "sqlfxdb - Get Database Configuration" in the *Administrative API Reference*) before they can be used by this utility. This can be a problem if the user does not have sufficient space in the *logpath*. The overflow log path is provided for this reason. During rollforward 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 rollforward 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 environment, the overflow log path can be relative if the server is local.

NumChngLgOvrflw
MPP only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified node only.

pChngLogOvrflw
MPP 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 node only.

ConnectMode
Input. Valid values (defined in `sqlutil`) are:

SQLUM_OFFLINE
Offline roll forward. This value must be specified for database rollforward recovery.

SQLUM_ONLINE
Online roll forward.

pTablespaceList
Input. A pointer to a structure containing the names of the table spaces to be rolled forward to the end-of-logs or to a specific point in time. If not specified, the table spaces needing rollforward will be selected.

AllNodeFlag
MPP only. Input. Indicates whether the rollforward operation is to be applied to all nodes defined in `db2nodes.cfg`. Valid values are:

SQLURF_NODE_LIST
Apply to nodes in a node list that is passed in *pNodeList*.

SQLURF_ALL_NODES
Apply to all nodes. *pNodeList* should be NULL. This is the default value.

SQLURF_ALL_EXCEPT
Apply to all nodes except those in a node list that is passed in *pNodeList*.

SQLURF_CAT_NODE_ONLY
Apply to the catalog node only. *pNodeList* should be NULL.

NumNodes
Input. Specifies the number of nodes in the *pNodeList* array.

pNodeList
Input. A pointer to an array of node numbers on which to perform rollforward recovery.

NumNodeInfo
Input. Defines the size of the output parameter *pNodeInfo*, which must be large enough to hold status information from each node that is being rolled forward. In a single-partition environment, this parameter should be set to 1. The value of this parameter should be same as the number of nodes for which this API is being called.

DlMode
Input. This parameter is not currently used, and should be set to zero.

pReportFile
Input. This parameter is not currently used, and should be set to NULL.

pDroppedTblID
Input. A string containing the ID of the dropped table whose recovery is being attempted.

pExportDir
Input. The directory into which the dropped table data will be exported.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

REXX API Syntax

```
ROLLFORWARD DATABASE database-alias [USING :value] [USER username USING password]
[rollforward_action_clause | load_recovery_action_clause]
where rollforward_action_clause stands for:
    { TO point-in-time [AND STOP] |
      {
        [TO END OF LOGS [AND STOP] | STOP | CANCEL | QUERY STATUS | PARM CHECK }
        [ON {:nodelist | ALL NODES [EXCEPT :nodelist]}]
      }
    }
    [TABLESPACE {ONLINE |:tablespacenames [ONLINE]} ]
    [OVERFLOW LOG PATH default-log-path [:logpaths]]
and load_recovery_action_clause stands for:
    LOAD RECOVERY { CONTINUE | DEVICE_TERMINATE | TERMINATE }
```

REXX API Parameters

database-alias
Alias of the database to be rolled forward.

value A compound REXX host variable containing the output values. In the following, XXX represents the host variable name:

XXX.0	Number of elements in the variable
XXX.1	The application ID
XXX.2	Number of replies received from nodes
XXX.2.1.1	First node number
XXX.2.1.2	First state information
XXX.2.1.3	First next archive file needed
XXX.2.1.4	First first archive file to be deleted
XXX.2.1.5	First last archive file to be deleted
XXX.2.1.6	First last commit time
XXX.2.2.1	Second node number
XXX.2.2.2	Second state information
XXX.2.2.3	Second next archive file needed
XXX.2.2.4	Second first archive file to be deleted
XXX.2.2.5	Second last archive file to be deleted
XXX.2.2.6	Second last commit time
XXX.2.3.x	and so on.

username
Identifies the user name under which the database is to be rolled forward.

password
The password used to authenticate the user name.

point-in-time
A time stamp in ISO format, *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes,seconds, microseconds), expressed in Coordinated Universal Time (UTC).

tablespacenames
A compound REXX host variable containing a list of table spaces to be rolled forward. In the following, XXX is the name of the host variable:

XXX.0	Number of table spaces to be rolled forward
XXX.1	First table space name
XXX.2	Second table space name
XXX.*x*	and so on.

default-log-path
The default overflow log path to be searched for archived logs during recovery

logpaths
A compound REXX host variable containing a list of alternate log paths to be searched for archived logs during recovery. In the following, XXX is the name of the host variable:

XXX.0	Number of changed overflow log paths
XXX.1.1	First node
XXX.1.2	First overflow log path
XXX.2.1	Second node
XXX.2.2	Second overflow log path
XXX.3.1	Third node
XXX.3.2	Third overflow log path
XXX.x.1	and so on.

nodelist
A compound REXX host variable containing a list of nodes. In the following, XXX is the name of the host variable:

XXX.0	Number of nodes
XXX.1	First node

XXX.2	Second node
XXX.x	and so on.

Data Structure: RFWD-INPUT

This structure is used to pass information to the "Rollforward Database API" on page 148.

Table 8. Fields in the RFWD-INPUT Structure

Field Name	Data Type	Description
VERSION	sqluint32	Rollforward version.
PDBALIAS	Pointer	Database alias.
CALLERACTION	UNSIGNED SHORT	Action.
PSTOPTIME	Pointer	Stop time.
PUSERNAME	Pointer	User name.
PPASSWORD	Pointer	Password.
POVERFLOWLOGPATH	Pointer	Overflow log path.
NUMCHNGLGOVRFLW	UNSIGNED SHORT	Number of changed overflow log paths (MPP only).
PCHNGLOGOVRFLW	Structure	Changed overflow log paths (MPP only).
CONNECTMODE	UNSIGNED SHORT	Connect mode.
PTABLESPACELIST	Structure	A pointer to a list of table space names. For information about this structure, see "Data Structure: SQLU-TABLESPACE-BKRST-LIST" on page 100.
ALLNODEFLAG	SHORT	All node flag.
NUMNODES	SHORT	Size of the node list.
PNODELIST	Pointer	List of node numbers.
NUMNODEINFO	SHORT	Size of *pNodeInfo* in "Data Structure: RFWD-OUTPUT" on page 160.
DLMODE	UNSIGNED SHORT	This parameter is not currently used.
PREPORTFILE	Pointer	This parameter is not currently used.
PDROPPEDTBLID	Pointer	A string containing the ID of the dropped table whose recovery is being attempted.
PEXPORTDIR	Pointer	The directory into which the dropped table data will be exported.
NODENUM	SQL_PDB_NODE_TYPE	Node number.
PATHLEN	UNSIGNED SHORT	Length of the new log path.
LOGPATH	CHAR(255)	New overflow log path.

Data Structure: RFWD-INPUT

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: RFWD-INPUT */
/* ... */
SQL_STRUCTURE rfwd_input
{
  sqluint32              version;
  char                   *pDbAlias;
  unsigned short         CallerAction;
  char                   *pStopTime;
  char                   *pUserName;
  char                   *pPassword;
  char                   *pOverflowLogPath;
  unsigned short         NumChngLgOvrflw;
  struct sqlurf_newlogpath *pChngLogOvrflw;
  unsigned short         ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  short                  AllNodeFlag;
  short                  NumNodes;
  SQL_PDB_NODE_TYPE      *pNodeList;
  short                  NumNodeInfo;
  unsigned short         DlMode;         /* This parameter is not */
                                         /*   currently used. */
  char                   *pReportFile;   /* This parameter is not */
                                         /*   currently used. */
  char                   *pDroppedTblID;
  char                   *pExportDir;
};
/* ... */

/* File: sqlutil.h */
/* Structure: SQLURF-NEWLOGPATH */
/* ... */
SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE      nodenum;
  unsigned short         pathlen;
  char                   logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-RFWD-INPUT.
    05 SQL-VERSION            PIC 9(9) COMP-5.
    05 SQL-DBALIAS            USAGE IS POINTER.
    05 SQL-CALLERACTION       PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-STOPTIME           USAGE IS POINTER.
    05 SQL-USERNAME           USAGE IS POINTER.
    05 SQL-PASSWORD           USAGE IS POINTER.
    05 SQL-OVERFLOWLOGPATH    USAGE IS POINTER.
    05 SQL-NUMCHANGE          PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-P-CHNG-LOG-OVRFLW  USAGE IS POINTER.
    05 SQL-CONNECTMODE        PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-P-TABLESPACE-LIST  USAGE IS POINTER.
    05 SQL-ALLNODEFLAG        PIC S9(4) COMP-5.
    05 SQL-NUMNODES           PIC S9(4) COMP-5.
    05 SQL-NODELIST           USAGE IS POINTER.
    05 SQL-NUMNODEINFO        PIC S9(4) COMP-5.
    05 SQL-DLMODE             PIC 9(4) COMP-5.  * This parameter is not
                                                *   currently used.
    05 SQL-REPORTFILE         USAGE IS POINTER. * This parameter is not
                                                *   currently used.
    05 SQL-DROPPEDTBLID       USAGE IS POINTER.
    05 SQL-EXPORTDIR       USAGE IS POINTER.
*

* File: sqlutil.cbl
01 SQLURF-NEWLOGPATH.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
    05 SQL-PATHLEN            PIC 9(4) COMP-5.
    05 SQL-LOGPATH            PIC X(254).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
*
```

Data Structure: RFWD-OUTPUT

This structure is used to pass information from the "Rollforward Database API" on page 148 .

Table 9. Fields in the RFWD-OUTPUT Structure

Field Name	Data Type	Description
PAPPLICATIONID	Pointer	The address of a buffer of length `SQLU_APPLID_LEN+1` (defined in `sqlutil`) to hold an application identifier returned from the API. This identifier can be used with the database system monitor APIs to monitor the application. If this information is not of interest, specify the NULL pointer. In a partitioned database environment, returns only the application identifier for the catalog node.
PNUMREPLIES	Pointer	Number of node replies received. Each node that replies fills in an *sqlurf_info* structure in *pNodeInfo*. In a single-partition environment, the value of this parameter is 1.
PNODEINFO	Structure	Node reply information. A user-defined array of *NumNodeInfo sqlurf_info* structures.

Table 10. Fields in the SQLURF-INFO Structure

Field Name	Data Type	Description
NODENUM	SQL_PDB_NODE_TYPE	Node number.
STATE	LONG	State information.
NEXTARCLOG	UNSIGNED CHAR(13)	A 12-byte buffer to hold the returned name of the next required archived log file. If a caller action other than `SQLUM_QUERY` is specified, the value returned in this field indicates that an error occurred when accessing the file. Possible causes are: • The file was not found in the database log directory, nor on the path specified by the overflow log path parameter. • The user exit program failed to return the archived file.

Table 10. Fields in the SQLURF-INFO Structure (continued)

Field Name	Data Type	Description
FIRSTARCDEL	UNSIGNED CHAR(13)	A 12-byte buffer to hold the returned name of the first archived log file that is no longer needed for recovery. This file, and all files up to and including *lastarcdel*, can be moved to make room on the disk. For example, if the values returned in *firstarcdel* and *lastarcdel* are `S0000001.LOG` and `S0000005.LOG`, the following log files can be moved: • `S0000001.LOG` • `S0000002.LOG` • `S0000003.LOG` • `S0000004.LOG` • `S0000005.LOG`
LASTARCDEL	UNSIGNED CHAR(13)	A 12-byte buffer to hold the returned name of the last archived log file that can be removed from the database log directory.
LASTCOMMIT	UNSIGNED CHAR(27)	A 26-character string containing a time stamp in ISO format. This value represents the time stamp of the last committed transaction after the rollforward operation terminates.

Possible values for *STATE* (defined in `sqlutil`) are:

SQLURFQ_NOT_AVAILABLE
Could not connect to the node.

SQLURFQ_NOT_RFW_PENDING
Database is not in rollforward pending state.

SQLURFQ_DB_RFW_PENDING
Database is in rollforward pending state.

SQLURFQ_TBL_RFW_PENDING
Table space is in rollforward pending state.

SQLURFQ_DB_RFW_IN_PROGRESS
Database is in rollforward-in-progress state.

SQLURFQ_TBL_RFW_IN_PROGRESS
Table space is in rollforward-in-progress state.

SQLURFQ_DB_RFW_STOPPING
Database rollforward operation was interrupted while processing a STOP request.

SQLURFQ_TBL_RFW_STOPPING
Table space rollforward operation was interrupted while processing a STOP request.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: RFWD-OUTPUT */
/* ... */
SQL_STRUCTURE rfwd_output
{
  char            *pApplicationId;
  long            *pNumReplies;
  struct sqlurf_info *pNodeInfo;
};
/* ... */

/* File: sqlutil.h */
/* Structure: SQLURF-INFO */
/* ... */
SQL_STRUCTURE sqlurf_info
{
  SQL_PDB_NODE_TYPE nodenum;
  long            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];
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-RFWD-OUTPUT.
    05 SQL-APPLID             USAGE IS POINTER.
    05 SQL-NUMREPLIES         USAGE IS POINTER.
    05 SQL-P-NODE-INFO        USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQLURF-INFO.
    05 SQL-NODENUM           PIC S9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-STATE             PIC S9(9) COMP-5.
    05 SQL-NEXTARCLOG        PIC X(12).
    05 FILLER                PIC X.
    05 SQL-FIRSTARCDEL       PIC X(12).
    05 FILLER                PIC X.
    05 SQL-LASTARCDEL        PIC X(12).
    05 FILLER                PIC X.
    05 SQL-LASTCOMMIT        PIC X(26).
    05 FILLER                PIC X.
    05 FILLER                PIC X(2).
*
```

Example Rollforward Sessions

CLP Examples

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected, before stopping it and possibly missing logs. This is especially important if a bad log is found during rollforward recovery, and the bad log is interpreted to mean the "end of logs". In such cases, an undamaged backup copy of that log could be used to continue the rollforward operation through more logs.

Example 2

Roll forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   overflow log path (/logs)
```

Example 5 (MPP)

There are three nodes: 0, 1, and 2. Table space TBS1 is defined on all nodes, and table space TBS2 is defined on nodes 0 and 2. After restoring the database on node 1, and TBS1 on nodes 0 and 2, roll the database forward on node 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 ("Database is recovered but one or more table spaces are off-line on node(s) 0 and 2.").

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

This rolls TBS1 forward on nodes 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 6 (MPP)

After restoring table space TBS1 on nodes 0 and 2 only, roll TBS1 forward on nodes 0 and 2:

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

Node 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on node 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on nodes (0, 2) tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on node 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the node clause is not accepted. The rollforward operation must take place on all the nodes on which the table space resides.

After restoring TBS1 on node 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (MPP)

After restoring a table space on all nodes, roll forward to PIT2, but do not specify AND STOP. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all nodes **

db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

Example 8 (MPP)

Rollforward recover a table space that resides on eight nodes (3 to 10) listed in the db2nodes.cfg file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The nodes on which the table space resides do not have to be specified. The utility defaults to the db2nodes.cfg file.

Example 9 (MPP)

Rollforward recover six small table spaces that reside on a single node nodegroup (on node 6):

```
db2 rollforward database dwtest to end of logs on node (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

A sample DB2 command script, and information on how to use it, are provided in "Appendix F. Recovery CLP Script" on page 425.

API Examples

Sample programs containing DB2 APIs and embedded SQL calls, and information on how to use them, are provided in "Appendix E. Recovery Sample Programs" on page 359.

Rollforward Restrictions

The following restrictions apply to the rollforward utility:

- You can only invoke one rollforward operation at a time. If there are many table spaces to recover, you can specify all of them in the same operation.
- If you have renamed a table space following the most recent backup operation, ensure that you use the new name when rolling the table space forward. The previous table space name will not be recognized.
- You cannot cancel a rollforward operation that is running. You can only cancel a rollforward operation that has completed, but for which the STOP option has not been specified, or a rollforward operation that has failed before completing.
- You cannot *continue* a table space rollforward operation to a point in time, specifying a time stamp that is less than the previous one. If a point in time is not specified, the previous one is used. You can initiate a rollforward operation to a point in time by just specifying STOP, but this is only allowed if the table spaces involved were all restored from the same offline backup image. In this case, no log processing is required. If you start another rollforward operation with a different table space list before the in-progress rollforward operation is either completed or cancelled, an error message (SQL4908) is returned. Invoke the LIST TABLESPACES command on all nodes to determine which table spaces are currently being rolled forward (rollforward in progress state), and which table spaces are ready to be rolled forward (rollforward pending state). You have three options:
  - Finish the in-progress rollforward operation on all table spaces.
  - Finish the in-progress rollforward operation on a subset of table spaces. (This may not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all nodes.)
  - Cancel the in-progress rollforward operation.

Troubleshooting Rollforward

Do not restore table spaces without cancelling a rollforward operation that is in progress; otherwise, you may have a table space set in which some table spaces are in rollforward in progress state, and some table spaces are in rollforward pending state. A rollforward operation that is in progress will only operate on the tables spaces that are in rollforward in progress state.

If the rollforward utility encounters a non-recoverable operation (for example, load with no copy), the associated table space is put in restore pending state. To remove the table space from restore pending state, you must restore from a more recent database- or table space-level backup image.

SQL1271 is returned (even if AND STOP has not yet been requested) as a warning that there is a table space in rollforward pending or restore pending

state. During a database rollforward operation, this means that one of the table spaces has been taken offline by the rollforward utility. During a *table space* rollforward operation, this can mean that one of the table spaces being rolled forward was taken offline by the rollforward utility, or that there is another (unlisted) table space that still needs to be rolled forward.

SQL1272 is returned if all table spaces that were being rolled forward (either specified in the list, or because they were in rollforward pending state), have been taken offline by the rollforward utility. This may mean that they contain a table that has undergone a non-recoverable load operation, or a table with NOT LOGGED INITIALLY processing. If this error is returned, the table space rollforward operation should have been stopped (that is, it should no longer be in progress).

Part 2. High Availability

Chapter 5. Introducing High Availability and Failover Support

Successful e-businesses depend on the uninterrupted availability of transaction processing systems, which in turn are driven by database management systems, such as DB2, that must be available 24 hours a day and 7 days a week ("24 x 7").

High Availability

High availability (HA) is the term that is used to describe systems that run and are available to customers more or less all the time. For this to occur:

- Transactions must be processed efficiently, without appreciable performance degradations (or even loss of availability) during peak operating periods. In a partitioned database environment, DB2 can take advantage of both intrapartition and interpartition parallelism to process transactions efficiently. *Intrapartition parallelism* can be used in an SMP environment to process the various components of a complex SQL statement simultaneously. *Interpartition parallelism* in a partitioned database environment, on the other hand, refers to the simultaneous processing of a query on all participating nodes; each node processes a subset of the rows in the table. For more information about parallelism, see the *Administration Guide* .
- Systems must be able to recover quickly when hardware or software failures occur, or when disaster strikes. Use of a journaled file system may be helpful. A *journaled file system* is a file system that automatically logs changes to it structure. This ensures that any changes to the file system structure are safe from system crashes, assuming that the storage medium is not lost or corrupted. After a system crash, the changes in these logs are applied to the file system, in the order that they were initially logged. Journaled file systems are characterized by fast recovery times. They are preferred for environments with large file systems, or when data integrity is particularly important.

  The ability to recover quickly depends critically on having a proven backup and recovery strategy in place. For more information about recovery strategies, see "Chapter 1. Developing a Good Backup and Recovery Strategy" on page 3 .
- Software that powers the enterprise databases must be continuously running and available for transaction processing. To keep the database manager running, you must ensure that another database manager can take

over if it fails. This is called failover. *Failover* capability allows for the automatic transfer of workload from one system to another when there is hardware failure.

Failover protection can be achieved by keeping a copy of your database on another machine that is perpetually rolling the log files forward. *Log shipping* is the process of copying whole log files to a standby machine, either from an archive device, or through a user exit program running against the primary database. With this approach, the primary database is restored to the standby machine, using either the DB2 restore utility or the split mirror function. You can use the new suspended I/O support to quickly initialize the new database (see "High Availability through Online Split Mirror and Suspended I/O Support" on page 173). The secondary database on the standby machine continuously rolls the log files forward. If the primary database fails, any remaining log files are copied over to the standby machine. After a rollforward to the end of the logs and stop operation, all clients are reconnected to the secondary database on the standby machine.

Failover support can also be provided through platform-specific software that you can add to your system. For example:

- High Availability Cluster Multi-Processing, Enhanced Scalability, for AIX.

  For detailed information about HACMP/ES, see "Chapter 6. High Availability on AIX" on page 177, or the white paper entitled "IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES", which is available from the "DB2 UDB and DB2 Connect Online Support" Web site (http://www.ibm.com/software/data/pubs/papers/).
- Microsoft Cluster Server, for Windows NT or Windows 2000.

  For detailed information about MSCS, see "Chapter 7. High Availability on the Windows Operating System" on page 221 .
- Sun Cluster, or VERITAS Cluster Server, for the Solaris Operating Environment.

  For information about Sun Cluster 2.x, see "Chapter 8. High Availability in the Solaris Operating Environment" on page 253 ; for information about Sun Cluster 3.0, see the white paper entitled "DB2 and High Availability on Sun Cluster 3.0", which is available from the "DB2 UDB and DB2 Connect Online Support" Web site (http://www.ibm.com/software/data/pubs/papers/). For information about VERITAS Cluster Server, see the white paper entitled "DB2 and High Availability on VERITAS Cluster Server", which is also available from the "DB2 UDB and DB2 Connect Online Support" Web site.
- Multi-Computer/ServiceGuard, for Hewlett-Packard.

  For detailed information about HP MC/ServiceGuard, see the white paper entitled "IBM DB2 EE v.7.1 Implementation and Certification With Hewlett-Packard's MC/ServiceGuard High Availability Software", which is

available from the "DB2 UDB and DB2 Connect Online Support" Web site (http://www.ibm.com/software/data/pubs/papers/).

Failover strategies are usually based on clusters of systems. A *cluster* is a group of connected systems that work together as a single system. Each processor is known as a node within the cluster. Clustering allows servers to back each other up when failures occur, by picking up the workload of the failed server.

IP address takeover (or IP takeover) is the ability to transfer a server IP address from one machine to another when a server goes down; to a client application, the two machines appear at different times to be the same server.

Failover software may use *heartbeat monitoring* or *keepalive packets* between systems to confirm availability. Heartbeat monitoring involves system services that maintain constant communication between all the nodes in a cluster. If a heartbeat is not detected, failover to a backup system starts. End users are usually not aware that a system has failed.

The two most common failover strategies on the market are known as *idle standby* and *mutual takeover*, although the configurations associated with these terms may also be associated with different terms that depend on the vendor:

Idle Standby
: In this configuration, one system is used to run a DB2 instance, and the second system is "idle", or in standby mode, ready to take over the instance if there is an operating system or hardware failure involving the first system. Overall system performance is not impacted, because the standby system is idle until needed.

Mutual Takeover
: In this configuration, each system is the designated backup for another system. Overall system performance may be impacted, because the backup system must do extra work following a failover: it must do its own work plus the work that was being done by the failed system.

Failover strategies can be used to failover an instance, a partition, or multiple logical nodes.

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 making a backup copy of the mirror.

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:

- For making a clone database

  A read-only clone of the primary database can be used, for example, to create reports.
- As a standby database
- As a backup image

This command can only be issued against the split-off mirror, and the split-off mirror must first run **db2inidb** before it can be used (see "db2inidb - Initialize a Mirrored Database" on page 313).

In a partitioned database environment, the **db2inidb** command must be run on every partition before the split image from any of the partitions can be used. The tool can be run on all partitions simultaneously.

Making a Clone Database

A database clone can represent an offline "backup" of the primary (live) database. You cannot, however, back up the cloned database, restore this image on the original system, and roll forward through log files produced on the original system.

To clone a database, follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
2. Use an appropriate operating system-level command to split the mirror from the primary database.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Attach to the mirrored database from another machine.
5. Start the database instance:
   ```
   db2start
   ```

6. Initialize the mirrored database as a clone of the primary database:

   ```
   db2inidb database_alias as snapshot
   ```

   Note: This command will roll back transactions that are in flight when the split occurs.

Using the Split Mirror as a Standby Database

As the mirrored (standby) database continually rolls forward through the logs, new logs that are being created by the primary database are continually fetched from the primary system. To use the 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 an appropriate operating system-level command to split the mirror from the primary database.
3. Resume I/O on the primary database:

   ```
   db2 set write resume for database
   ```

4. Attach the mirrored database to another instance.
5. Put the mirrored database in rollforward pending state:

   ```
   db2inidb database_alias as standby
   ```

   If you have DMS table spaces (database managed space), you can take a full database backup to offload the overhead of taking a backup on the production database.
6. Set up a user exit program to retrieve the most recent log files from the primary system.
7. Roll the database forward to the end of the logs.
8. Continue retrieving log files, and rolling the database forward to the end of the logs until the primary database goes down.

Using the Split Mirror as a Backup Image

To use the 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 an appropriate operating system-level command to split the mirror from the primary database.
3. Resume I/O on the primary database:

   ```
   db2 set write resume for database
   ```

4. Use operating system-level commands to copy the mirrored data and logs over the primary system.
5. Start the database instance:

   ```
   db2start
   ```

6. Initialize the mirrored database as a "backup image" that can be used to copy the data on split-off disks back to the disks on the original system. (Do not bring back the file system that contains the log files, because the logs will be needed during the rollforward process.)

   ```
   db2inidb database_alias as mirror
   ```

7. Roll the database (on the original system) forward to the end of the logs.

Chapter 6. High Availability on AIX

Enhanced Scalability (ES) is a feature of High Availability Cluster Multi-Processing (HACMP) for AIX. This feature provides the same failover recovery and has the same event structure as HACMP, (see *HACMP for AIX, V4.2.2, Enhanced Scalability Installation and Administration Guide*). 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.

  For more information about user-defined events and the rules file, see "HACMP ES Event Monitoring and User-defined Events" on page 197.
- HACMP client utilities for monitoring and detecting status changes (in one or more clusters) from AIX physical nodes outside of the HACMP cluster.

The nodes in HACMP ES clusters exchange messages called *heartbeats*, or *keepalive packets*, by which each node informs the other nodes about its availability. A node that has stopped responding causes the remaining nodes in the cluster to invoke recovery. The recovery process is called a *node_down event* and may also be referred to as *failover*. The completion of the recovery process is followed by the re-integration of the node into the cluster. This is called a *node_up event*.

There are two types of events: standard events that are anticipated within the operations of HACMP ES, and user-defined events that are associated with the monitoring of parameters in hardware and software components.

One of the standard events is the node_down event. When planning what should be done as part of the recovery process, HACMP allows two failover options: hot (or idle) standby, and mutual takeover.

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, DB2 Universal Database Enterprise - Extended Edition (UDB EEE) runs in mutual takeover mode with partitions on each node. One exception is a scenario in which the catalog node is part of a hot standby configuration.

When planning a large DB2 installation on an RS/6000 SP using HACMP ES, you need to consider how to divide the nodes of the cluster within or between the RS/6000 SP frames. Having a node and its backup in different SP frames allows takeover in the event one frame goes down (that is, the frame power/switch board fails). However, such failures are expected to be exceedingly rare, because there are *N*+1 power supplies in each SP frame, and each SP switch has redundant paths, along with *N*+1 fans and power. In the case of a frame failure, manual intervention may be required to recover the remaining frames. This recovery procedure is documented in the SP Administration Guide. HACMP ES provides for recovery of SP node failures; recovery of frame failures is dependent on the proper layout of clusters within one or more SP frames.

Another planning consideration is how to manage big clusters. It is easier to manage a small cluster than a big one; however, it is also easier to manage one big cluster than many smaller ones. When planning, consider how your applications will be used in your cluster environment. If there is a single, large, homogeneous application running, for example, on 16 nodes, it is probably easier to manage the configuration as a single cluster rather than as eight two-node clusters. If the same 16 nodes contain many different applications with different networks, disks, and node relationships, it is probably better to group the nodes into smaller clusters. Keep in mind that nodes integrate into an HACMP cluster one at a time; it will be faster to start a configuration of multiple clusters rather than one large cluster. HACMP ES supports both single and multiple clusters, as long as a node and its backup are in the same cluster.

HACMP ES failover recovery allows pre-defined (also known as *cascading*) assignment of a resource group to a physical node. The failover recovery procedure also allows floating (or *rotating*) assignment of a resource group to a physical node. IP addresses, and external disk volume groups, or file systems, or NFS file systems, and application servers within each resource group specify either an application or an application component, which can be manipulated by HACMP ES between physical nodes by failover and

reintegration. Failover and reintegration behavior is specified by the type of resource group created, and by the number of nodes placed in the resource group.

For example, consider a DB2 database partition (logical node). If its log and table space containers were placed on external disks, and other nodes were linked to those disks, it would be possible for those other nodes to access these disks and to restart the database partition (on a takeover node). It is this type of operation that is automated by HACMP. HACMP ES can also be used to recover NFS file systems used by DB2 instance main user directories.

Read the HACMP ES documentation thoroughly as part of your planning for recovery with DB2 UDB EEE. 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). This is a starting point for completing the HACMP worksheets that are included in the documentation.

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. For DB2 UDB EEE, this means that all container names must be unique across all databases.

One way to achieve uniqueness is to include the partition number as part of the name. You can specify a node expression for container string syntax when creating either SMS or DMS containers. When you specify the expression, the node number can be part of the container name or, if you specify additional arguments, the results of those arguments can be part of the container name. Use the argument " $N" ([blank]$N) to indicate the node expression. The argument must occur at the end of the container string, and can only be used in one of the following forms:

Table 11. 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
```

Figure 18 on page 181 and Figure 19 on page 182 show an example of a DB2 SSA I/O subsystem configuration, and some of the planning necessary to

ensure both a highly available external disk configuration, and the ability to access all volume groups without conflict.

DB2 SSA I/O Subsystem Configuration - No single point of failure

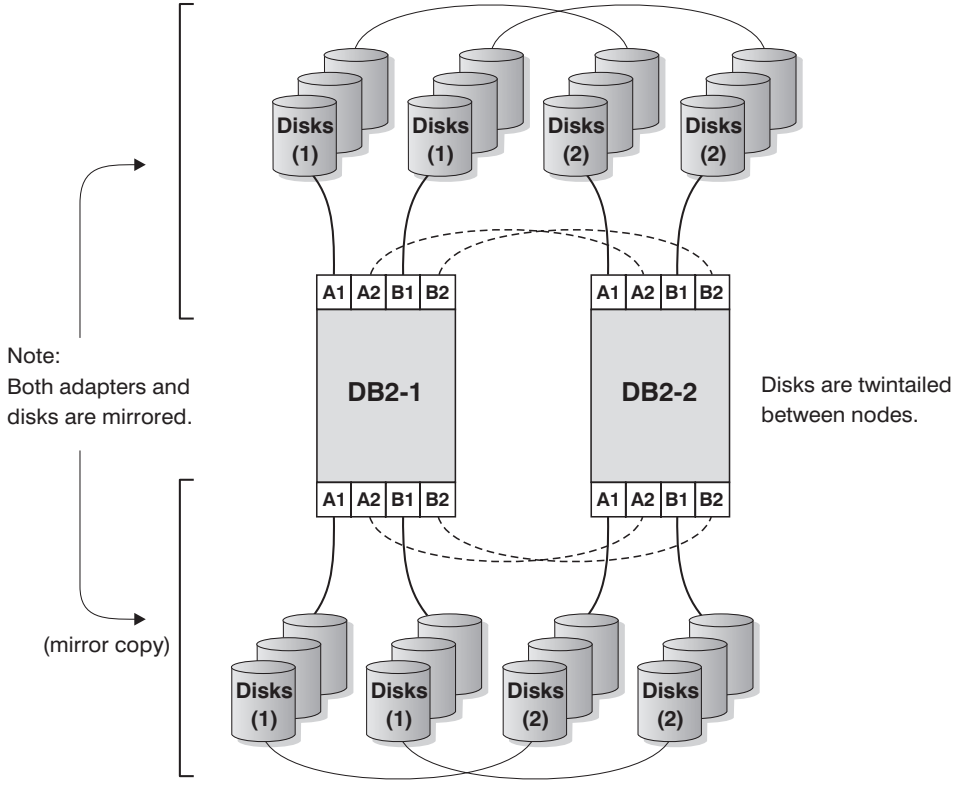


Figure 18. No Single Point of Failure

**DB2 SSA I/O Subsystem Configuration -
Volume group and logical volume setup**

db2 database testdata on filesystem /database instance name powertp

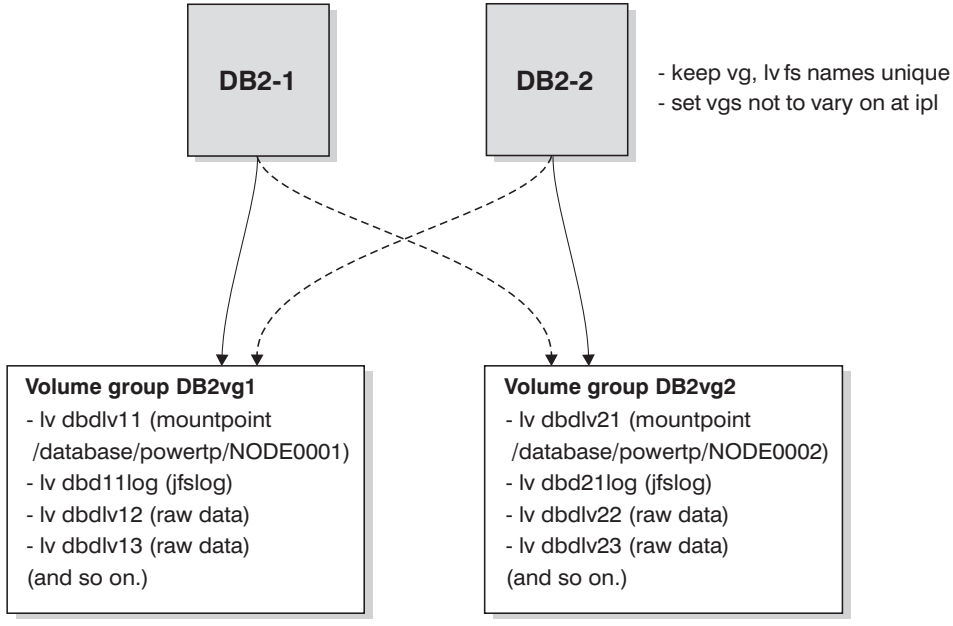


Figure 19. Volume Group and Logical Volume Setup

Configuring a DB2 Database Partition

Once configured, each database partition in an instance is started by HACMP ES, one physical node at a time. Multiple clusters are recommended for starting parallel DB2 configurations that are larger than four nodes. Note that in a 64-node parallel DB2 configuration, it is faster to start 32 two-node HACMP clusters in parallel, than four 16-node clusters.

A script file, `rc.db2pe`, is packaged with DB2 UDB EEE (and installed on each node in `/usr/bin`) to assist in configuring for HACMP ES failover or recovery in either hot standby or mutual takeover nodes. In addition, DB2 buffer pool sizes can be customized during failover in mutual takeover configurations from within `rc.db2pe`. (Buffer pool sizes need to be configured to ensure proper performance when two database partitions run on one physical node.)

When you create an application server in an HACMP configuration of a DB2 database partition, specify rc.db2pe as a start and stop script as follows:

```
/usr/bin/rc.db2pe <instance> <dpn> <secondary dpn> start <use switch>
/usr/bin/rc.db2pe <instance> <dpn> <secondary dpn> stop <use switch>

where:

<instance> is the instance name.
<dpn> is the database partition number.
<secondary dpn> is the "companion" database partition number in
   mutual takeover configurations only; in hot standby configurations,
   it is the same as <dpn>.
<use switch> is usually blank; when blank, it indicates that
   the SP switch network is used for the hostname field
   in the db2nodes.cfg file (all traffic for DB2 is routed over the SP switch);
   if not blank, the name used is the host name of the SP node to be used.
```

The DB2 command LIST DATABASE DIRECTORY is used from within rc.db2pe to find all databases configured for this database partition. The script file then looks for the /usr/bin/reg.parms.*DATABASE* file and the /usr/bin/failover.parms.*DATABASE* file, where *DATABASE* is each of the databases configured for this database partition. In a mutual takeover configuration, it is recommended that you create the parameter files reg.parms.*xxx* and failover.parms.*xxx*. In the failover.parms.*xxx* file, the settings for BUFFPAGE, DBHEAP, and any others affecting buffer pools, should be adjusted to account for the possibility of more than one buffer pool. Sample files reg.parms.SAMPLE and failover.parms.SAMPLE are provided for your use.

One of the important parameters in this environment is the *start_stop_time* database manager configuration parameter, which has a default value of 10 minutes. However, rc.db2pe sets this parameter to 2 minutes. You should set this parameter through rc.db2pe to a value of 10 minutes, or slightly more. In this context, the specified duration is the time interval between the failure of the partition, and its recovery. If applications running on a partition are issuing frequent COMMITs, 10 minutes following failure on a database partition should be sufficient time to roll back uncommitted transactions and to reach a point of consistency in the database on that partition. If your workload is heavy, or you have many partitions, you may need to increase the duration to decrease the probability of timeouts occurring before the rollback operation completes.

Following is an example of a hot standby configuration and a mutual takeover configuration. In both examples, the resource groups contain a Service IP switch alias address. This switch alias address is used for:

- NFS access to a file server for the DB2 instance owner file systems

- Other client access that needs to be maintained in the case of a failover, TSM (Tivoli Storage Manager, formerly ADSM) connection, or other similar operation.

If your implementation does not require these aliases, they can be removed. If removed, be sure to set the *MOUNT_NFS* parameter to `NO` in the `rc.db2pe` script file.

Example of a Hot Standby Configuration

The assumption in this example is that a hot standby configuration exists between physical nodes 1 and 2, and that the DB2 instance name is POWERTP. The database partition is 1, and the database is TESTDATA, residing on file system `/database`.

```
Resource group name: db2_dp_1
Node Relationship: cascading
Participating nodenames: node1_eth, node2_eth
Service_IP_label: nfs_switch_1     (<<< this is the switch alias address)
Filesystems: /database/powertp/NODE0001
Volume Groups: DB2vg1
Application Servers: db2_dp1_app
Application Server Start Script: /usr/bin/rc.db2pe powertp 1 1 start
Application Server Stop Script: /usr/bin/rc.db2pe powertp 1 1 stop
```

Example of a Mutual Takeover Configuration

The assumption in this example is that a mutual takeover configuration exists between physical nodes 1 and 2, and that the DB2 instance name is POWERTP. The database partitions are 1 and 2, and the database is TESTDATA, residing on file system `/database`.

```
Resource group name: db2_dp_1
Node Relationship: cascading
Participating nodenames: node1_eth, node2_eth
Service_IP_label: nfs_switch_1     (<<< this is the switch alias address)
Filesystems: /database/powertp/NODE0001
Volume Groups: DB2vg1
Application Servers: db2_dp1_app
Application Server Start Script: /usr/bin/rc.db2pe powertp 1 2 start
Application Server Stop Script: /usr/bin/rc.db2pe powertp 1 2 stop

Resource group name: db2_pd_2
Node Relationship: cascading
Participating nodenames: node2_eth, node1_eth
Service_IP_label: nfs_switch_2     (<<< this is the switch alias address)
Filesystems: /database/powertp/NODE0002
Volume Groups: DB2vg2
Application Servers: db2_dp2_app
Application Server Start Script: /usr/bin/rc.db2pe powertp 2 1 start
Application Server Stop Script: /usr/bin/rc.db2pe powertp 2 1 stop
```

Configuration of an NFS Server Node

The `rc.db2pe` script can also be used to make available NFS-mounted directories of DB2 parallel instance user directories. This can be accomplished

by setting the *MOUNT_NFS* parameter to YES in the rc.db2pe script file, and configuring the NFS failover server pair as follows:

- Configure the home directory and export it as "root" using /etc/exports and the **exportfs** command to the IP address used on the nodes in the same subnet as the NFS server's IP address. Include both the HACMP boot and service addresses. The NFS server's IP address is the same address as the service address in HACMP, and which can be taken over by a backup. The home directory of the DB2 instance owner should be NFS-mounted directly, not automounted. (The use of the automounter is not supported by the scripts as a DB2 instance owner home directory.)
- Using SMIT or a bottom-line configuration, create a separate /etc/filesystems entry for this file system, so that all nodes in the DB2 parallel grouping, including the file server, can mount using the NFS file system command.

  For example, an /nfshome JFS file system can be exported to all nodes as /dbhome. Each node creates an NFS file system /dbname, which is nfs_server:/nfshome. Therefore, the home directory of the DB2 instance owner would be /dbhome/powertp if the instance name is "powertp".

  Ensure that the NFS parameters for the mount in /etc/filesystems are "hard", "bg", "intr", and "rw".
- Ensure that the DB2 instance owner definitions associated with the home directory /dbhome/powertp in /etc/passwd are the same on all nodes.

  The user definitions in an SP environment are typically created on the control workstation, and "supper" or "pcp" is used to distribute /etc/passwd, /etc/security/passwd, /etc/security/user, and /etc/security/group to all nodes.
- Do *not* configure the "nfs_filesystems to export" in HACMP resource groups for the volume group and the file system that is exported. Instead, configure it normally to NFS. The scripts for the NFS server will control the exporting of the file systems.
- Ensure that the major number of the volume group where the file system resides is the same on both the primary node and the takeover node. This is accomplished by using **importvg** with the -V option.
- Verify that the *MOUNT_NFS* parameter is set to YES in the rc.db2pe script file, and that each node has the NFS file system to mount in /etc/filesystems. If this is not the case, rc.db2pe will not be able to mount the file system and start DB2.
- If the DB2 instance owner was already created, and you are copying the user's directory structure to the file system you are creating, ensure that you **tar** (-cvf) the directory. This ensures the preservation of symbolic links.
- Do not forget to mirror both the adapters and the disks for the logical volumes, and the file system logs of the file system you are creating.

Example of an NFS Server Takeover Configuration

The assumption in this example is that there is an NFS server file system /nfshome in the volume group nfsvg over the IP address "nfs_server". The DB2 instance name is POWERTP, and the home directory is /dbhome/powertp.

```
Resource group name: nfs_server
Node Relationship: cascading
Participating nodenames: node1_eth, node2_eth
Service_IP_label: nfs_server     (<<< this is the switch alias address)
Filesystems: /nfshome
Volume Groups: nfsvg
Application Servers: nfs_server_app
Application Server Start Script: /usr/bin/rc.db2pe powertp NFS SERVER start
Application Server Stop Script: /usr/bin/rc.db2pe powertp NFS SERVER stop
```

In this example:

- /etc/filesystems on all nodes would contain an entry for /dbhome as mounting nfs_server:/nfshome. nfs_server is a Service IP switch alias address.
- /etc/exports on the nfs_server node and the backup node would include the boot and service addresses, and contain an entry for /nfsfs -root=nfs_switch_1, nfs_switch_2, ...

Considerations When Configuring the SP Switch

When implementing HACMP ES with the SP switch, consider the following:

- There are "base" and "alias" addresses on the SP switch. The base addresses are those defined in the SP System Data Repository (SDR), and are configured by rc.switch when the system is "booted". The alias addresses are IP addresses configured, in addition to the base address, into the css0 interface through the **ifconfig** command with an alias attribute. For example:

  ```
  ifconfig css0 inet alias sw_alias_1 up
  ```

- When configuring the DB2 db2nodes.cfg file, SP switch "base" IP address names should be used for both the "hostname" and the "netname" field. Switch IP address aliases are *only* used to maintain NFS connectivity. DB2 failover is achieved by restarting DB2 with the **db2start** (RESTART) command (which updates db2nodes.cfg).
- Do not confuse the switch addresses with the etc/hosts aliases. Both the SP switch addresses and the SP switch alias addresses are real in either etc/hosts or DNS. The switch alias addresses are not another name for the SP switch base address; each has its own separate address.
- The SP switch base addresses are always present on a node when it is up. HACMP ES does not configure or move these addresses between nodes.
- If you intend to use SP switch alias addresses, configure these to HACMP as boot and service addresses for "heartbeating" and IP address takeover. If you do not intend to use SP switch alias addresses, configure the base SP

switch address to HACMP as a service address for ″heartbeating″ *only* (no IP address takeover). Do not, in any configuration, configure alias addresses *and* the switch base address; this configuration is not supported by HACMP ES.

- Only the SP switch alias addresses (and not the SP switch base addresses) are moved between nodes for an IP takeover configuration.
- The need for SP switch aliases arises because there can only be one SP switch adapter per node. Using alias addresses allows a node to take over another node's switch alias IP address without adding another switch adapter. This is useful in nodes that are ″slot-constrained″. For more information about handling recovery from SP switch adapter failures, see the network failure section under "HACMP ES Script Files" on page 201.
- If you configure the SP switch for IP address takeover, you will need to create two extra alias IP addresses per node: one as a boot address and one as a service address.
- Do not forget to use ″HPS″ in the HACMP ES network name definition for an SP switch base IP address or an SP switch alias IP address.
- `rc.cluster` in HACMP automatically **ifconfig**s in the SP switch boot address when HACMP is started. No additional configuration is required, other than creating the IP address and name, and defining them to HACMP.
- The Eprimary node of the SP switch is the server that implements the Estart, Efence, and Eunfence commands. The HACMP scripts attempt to Eunfence or to Estart a node when HACMP is started, and make the switch available if it is defined as one of its networks. For this reason, ensure that the Eprimary node is available when you start HACMP. The HACMP code waits up to 12 minutes for an Eprimary failover to complete before it exits with an error.
- The Eprimary node of the SP switch is moved between nodes by the SP Parallel System Support Program (PSSP), and not HACMP. If an Eprimary node goes offline, the PSSP automatically has a backup node assume responsibility as the Eprimary node. The switch network is unaffected by this change and remains up.

DB2 HACMP Configuration Examples

The following examples illustrate different failover support configurations and show what happens when failure occurs.

In the case of DB2 HACMP mutual takeover configurations (Figure 20 on page 189, Figure 21 on page 190, and Figure 22 on page 191):

- HACMP adapters are defined for ethernet, and SP switch alias boot and service aliases — base addresses are untouched. Remember to use an ″HPS″ string in the HACMP network name.

- The NFS_server/nfshome is mounted as /dbhome on all nodes through switch aliases.
- The db2nodes.cfg file contains SP switch base addresses. The db2nodes.cfg file is changed by the **db2start** (RESTART) command after a DB2 database partition (logical node) failover.
- The SP switch alias boot addresses are not shown.
- Nodes can be in different SP frames.

DB2 HACMP Mutual Takeover with NFS Failover - Normal

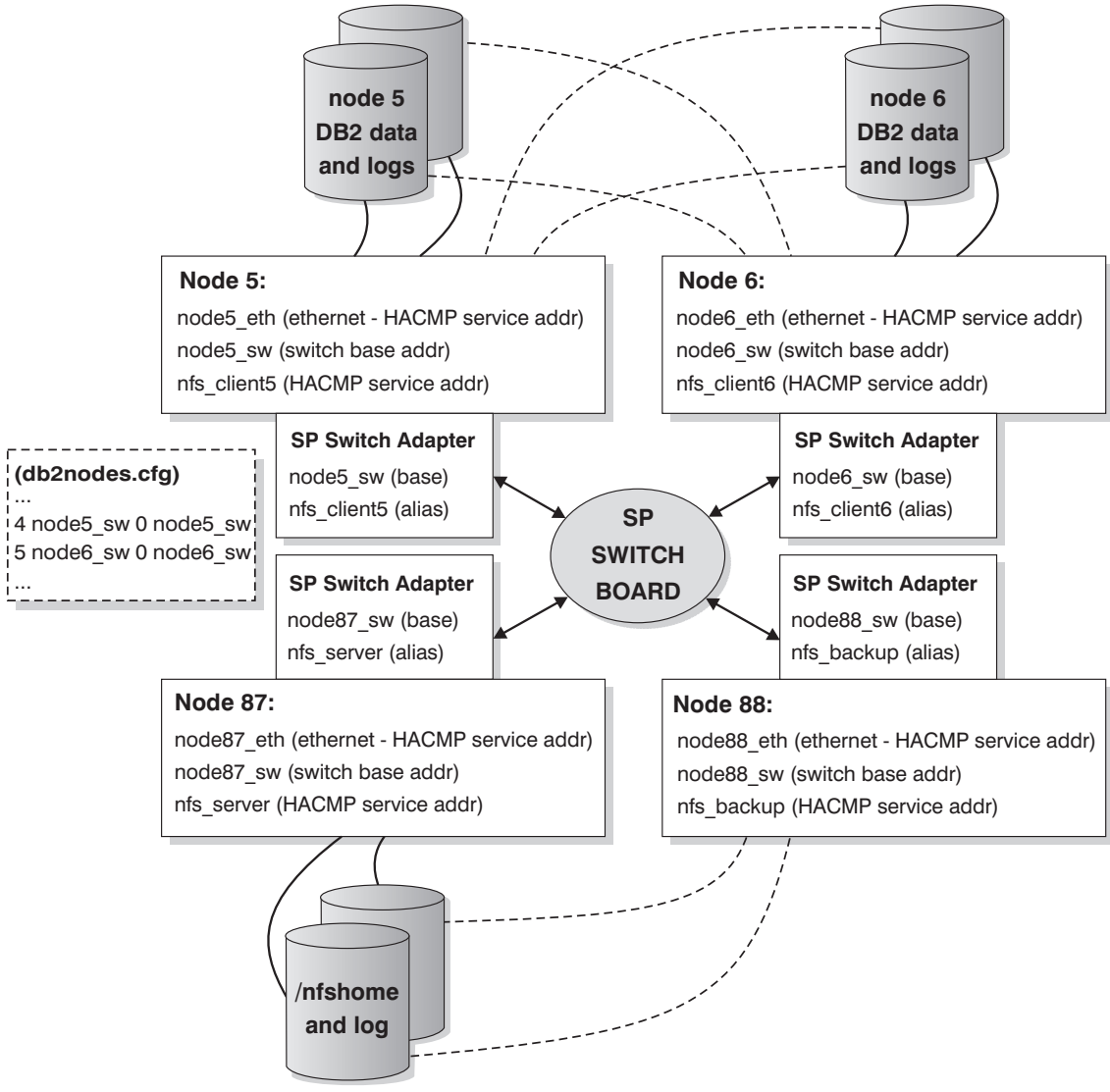


Figure 20. Mutual Takeover with NFS Failover - Normal

DB2 HACMP Mutual Takeover with NFS Failover - NFS failover

- nfs_server SP Switch alias IP addr and nfs mounted /nfshome moved from node 87 to 88.
- SP switch arp code has functionality to update all switch arp caches with this move.

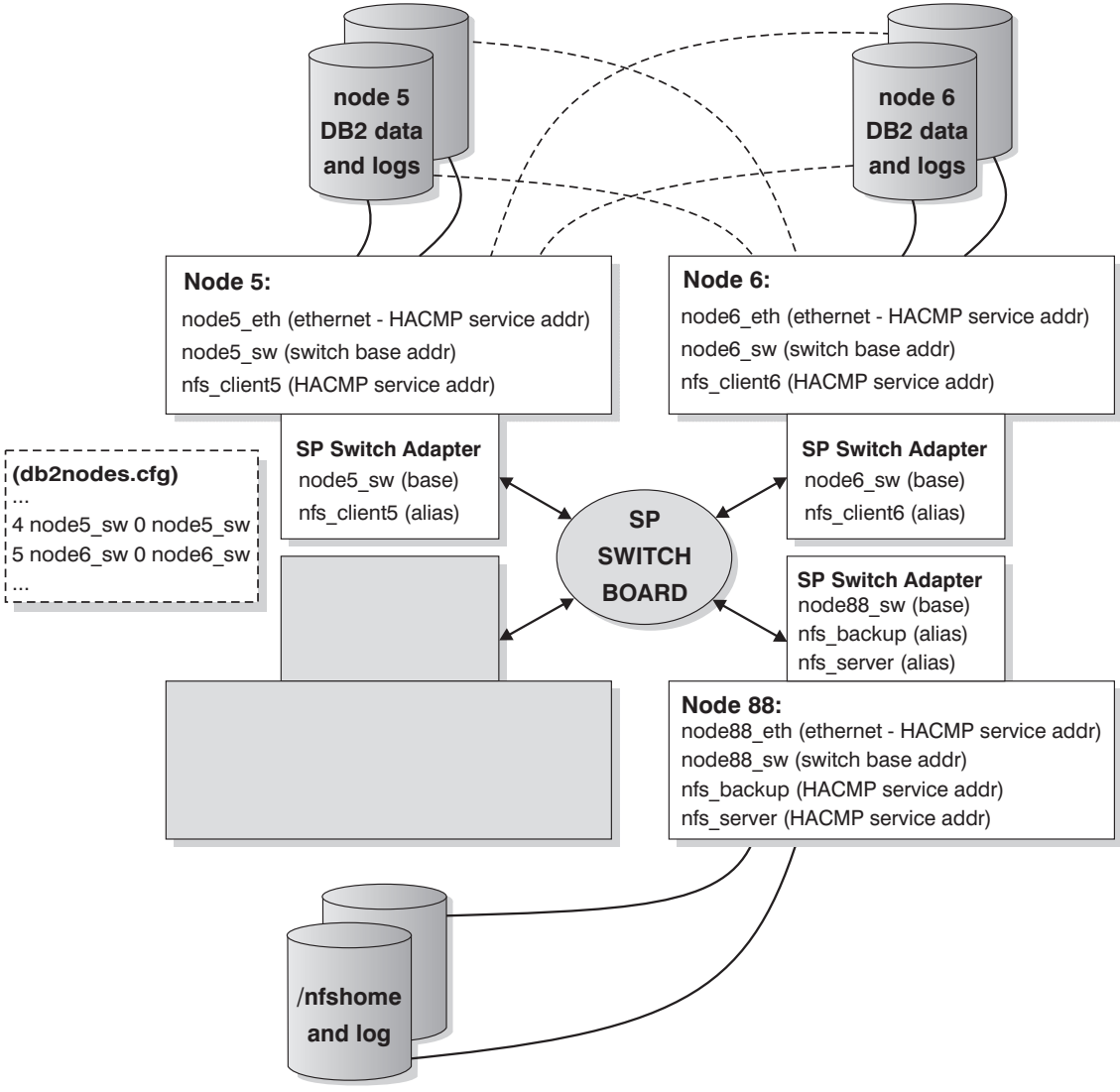


Figure 21. Mutual Takeover with NFS Failover - NFS Failover

DB2 HACMP Mutual Takeover with NFS Failover - DB2 failover

- switch IP address takeover allows other servers (like ADSM) to retain connectivity.
- Node 5 runs 2 logical nodes of DB2.

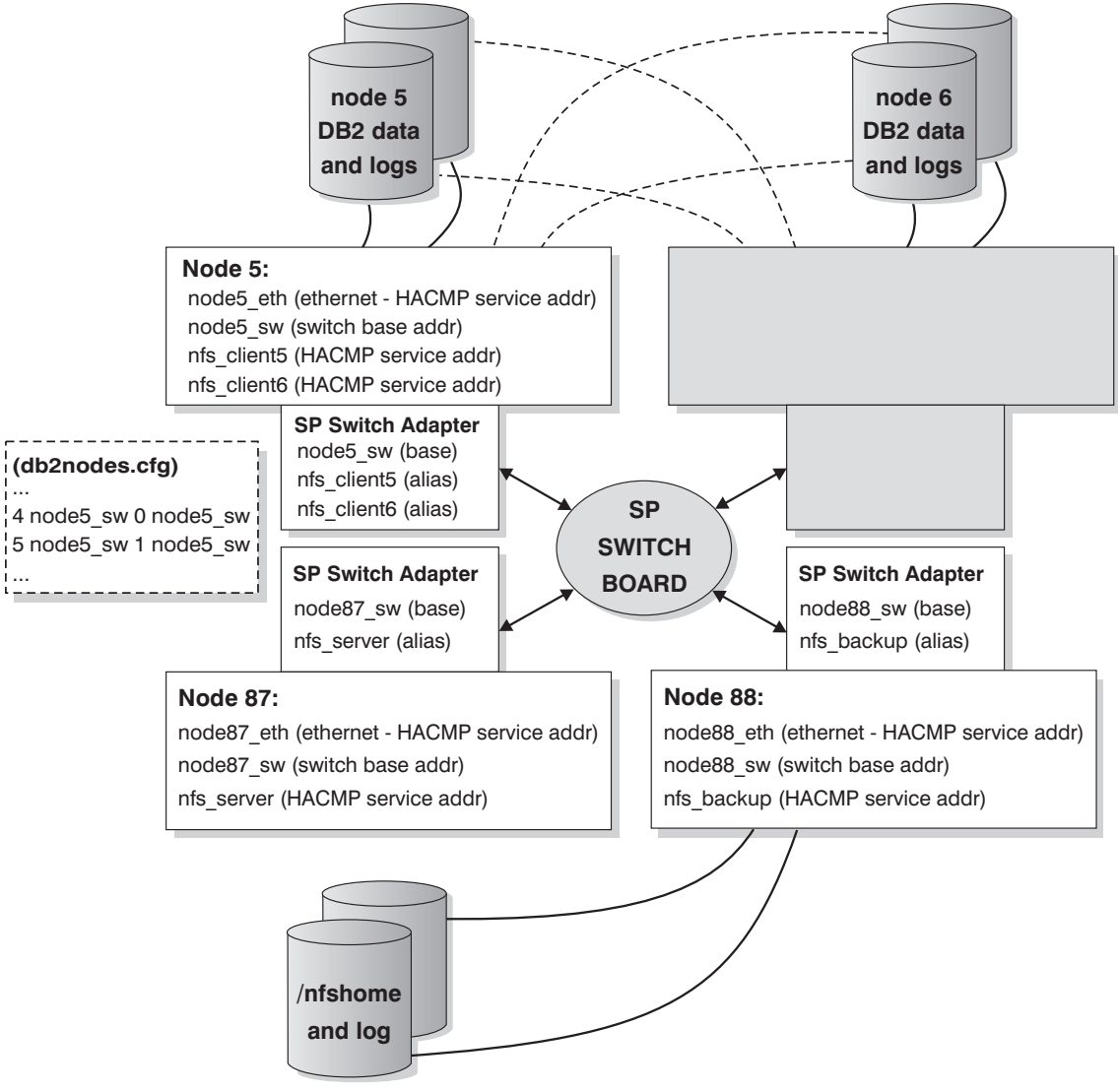


Figure 22. Mutual Takeover with NFS Failover - DB2 Failover

In the case of DB2 HACMP hot standby configurations (Figure 23 on page 193 and Figure 24 on page 194):

- HACMP adapters are defined for ethernet, and SP switch alias boot and service aliases — base addresses are untouched. Remember to use an ″HPS″ string in the HACMP network name.
- The `NFS_server/nfshome` is mounted as `/dbhome` on all nodes through switch aliases.
- The `db2nodes.cfg` file contains SP switch base addresses. The `db2nodes.cfg` file is changed by the **db2start** (RESTART) command after a DB2 database partition (logical node) failover.
- The SP switch alias boot addresses are not shown.

DB2 HACMP Hot Standby with NFS Failover - Normal

Note: Hot Standby node can back up more than one node, depending on disk cabling.

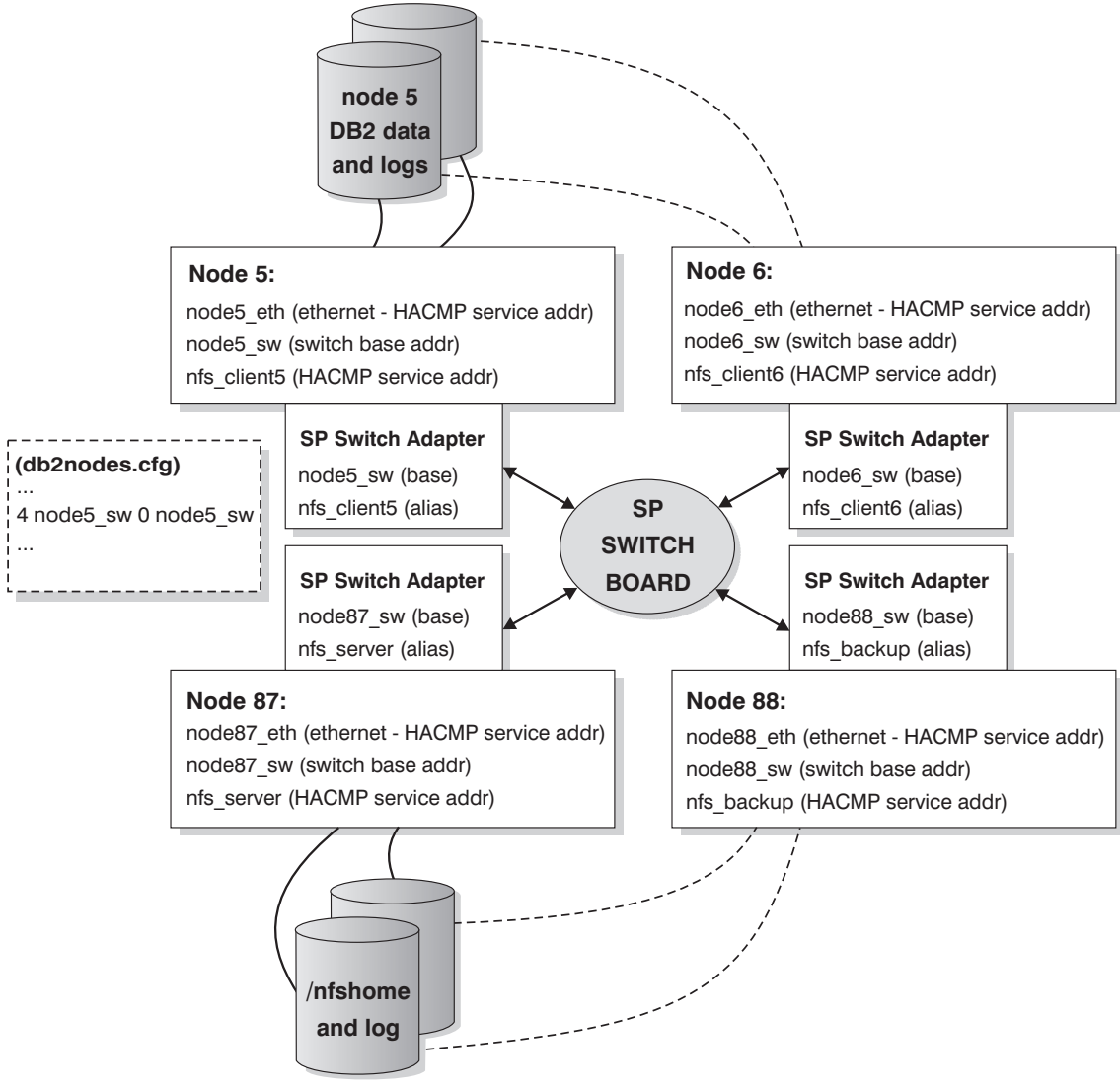


Figure 23. Hot Standby with NFS Failover - Normal

DB2 HACMP Hot Standby with NFS Failover- DB2 Failover

Note: Hot Standby node can back up more than one node, depending on disk cabling.

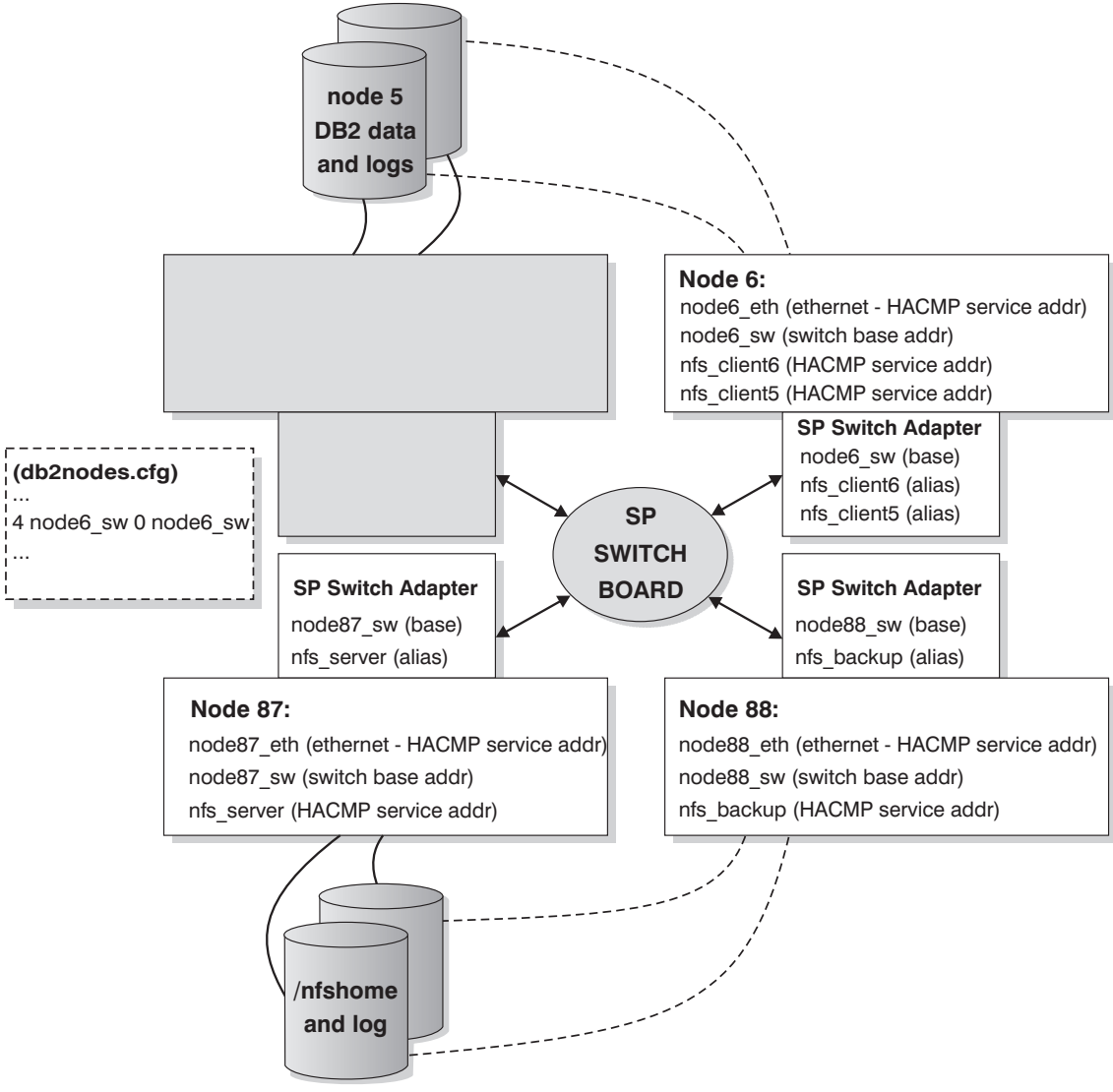


Figure 24. Hot Standby with NFS Failover - DB2 Failover

In the case of DB2 HACMP mutual takeover without NFS failover configurations (Figure 25 on page 195 and Figure 26 on page 196):

- HACMP adapters are defined for ethernet, and SP switch base addresses. Remember that when base addresses are configured to HACMP as service

addresses, there is no boot address (only a ″heartbeat″). Do not forget to use an ″HPS″ string in the HACMP network name for the SP switch.

- The `db2nodes.cfg` file contains SP switch base addresses. The `db2nodes.cfg` file is changed by the **db2start** (RESTART) command after a DB2 database partition (logical node) failover.
- No NFS failover functions are shown.
- Nodes can be in different SP frames.

DB2 HACMP Mutual Takeover without NFS Failover - Normal

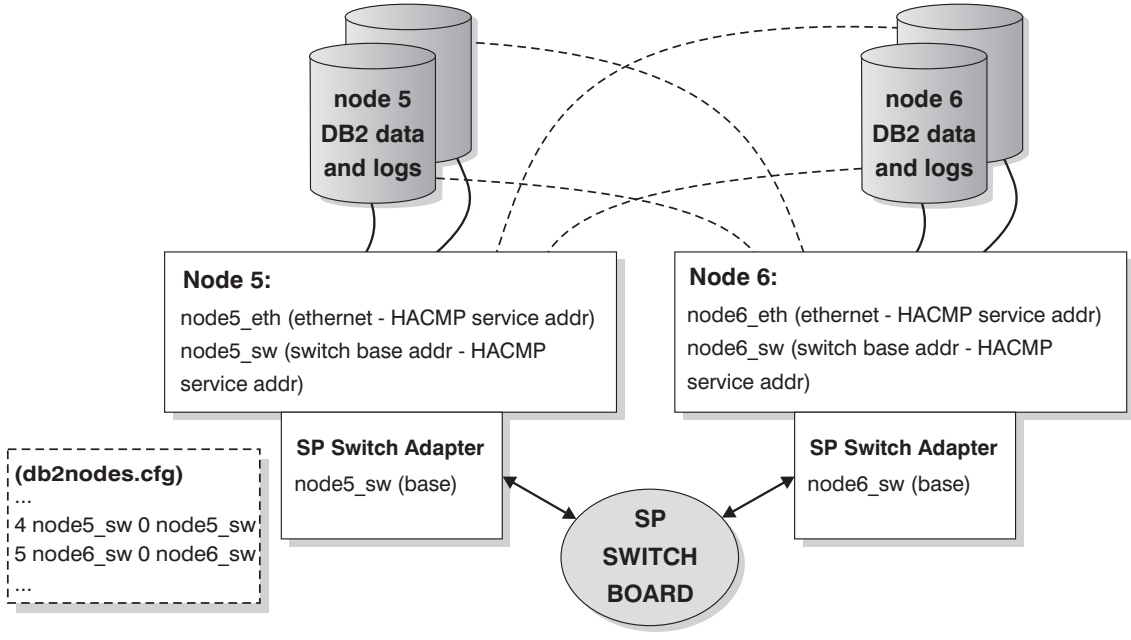


Figure 25. Mutual Takeover without NFS Failover - Normal

DB2 HACMP Mutual Takeover without NFS Failover - DB2 failover

- Node 5 runs 2 logical nodes of DB2.

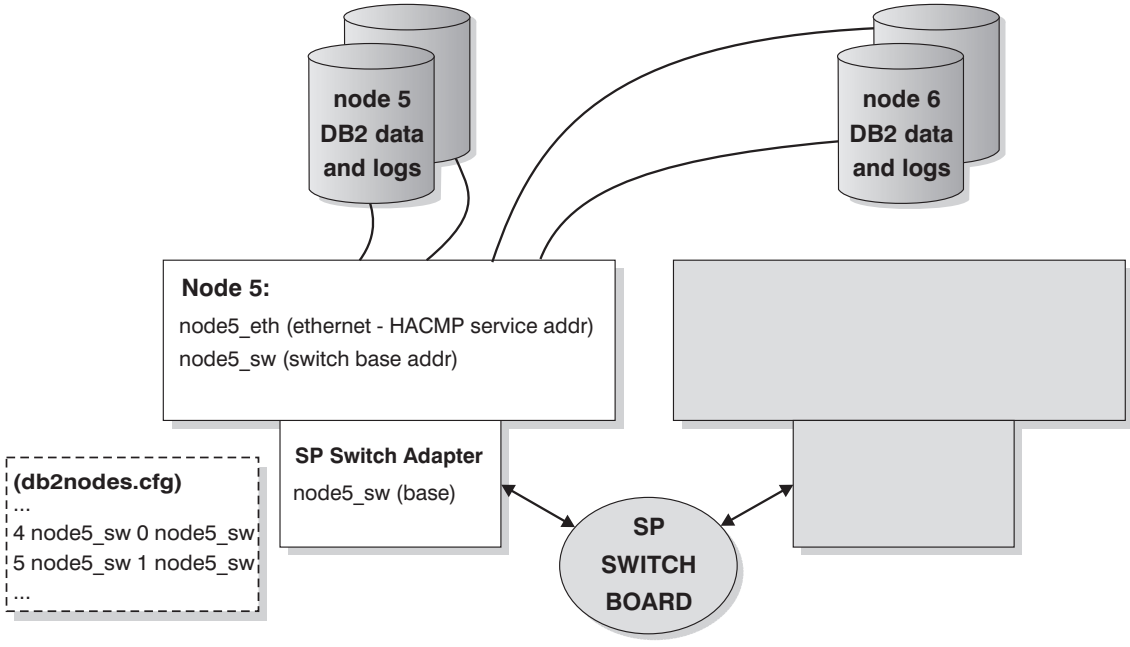


Figure 26. Mutual Takeover without NFS Failover - DB2 Failover

DB2 HACMP Startup Recommendations

It is recommended that you do not specify that HACMP is to be started at boot time in `/etc/inittab`. HACMP should be started manually after the nodes are booted. This allows for non-disruptive maintenance of a failed node.

As an example of "disruptive maintenance", consider the case in which a node has a hardware failure and crashes. Failover is initiated automatically by HACMP, and recovery completes successfully. However, the failed node needs to be fixed. If HACMP was configured in `/etc/inittab` to start on reboot, this node will try to reintegrate after boot completion, which is not desirable in this case.

For "non-disruptive maintenance", consider manually starting HACMP on each node. In this way, failed nodes can be fixed and reintegrated without affecting the other nodes. The `ha_cmd` script is provided for controlling HACMP start and stop commands from the control workstation.

Note: When creating a DB2 instance for the first time, the following entry is appended to the /etc/inittab file:

```
rcdb2:2:once:/etc/rc.db2 > /dev/console 2>&1 # Autostart DB2 Services
```

If HACMP or HACMP ES is enabled, update the /etc/inittab file by placing the above line before the HACMP entry. Following is a sample HACMP entry in the /etc/inittab file:

```
clinit:a:wait:touch /usr/sbin/cluster/.telinit # HACMP for AIX
```

The entry must be the last entry in the /etc/inittab file.

HACMP ES Event Monitoring and User-defined Events

Shutting down DB2 database partitions on an AIX physical node when paging space reaches a certain percentage of fullness, and restarting a DB2 database partition, or initiating a failover operation if a process dies on a given node, are two examples of user-defined events. 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 subdirectory.

A rules file, /user/sbin/cluster/events/rules.hacmprd, contains HACMP events. Each event description in this file has the following nine components:

- Event name, which must be unique.
- State, or qualifier for the event. The event name and state are the rule triggers. HACMP ES Cluster Manager initiates recovery only if it finds a rule with a trigger corresponding to the event name and state.
- Resource program path, a full-path specification of the *xxx*.rp file containing the recovery program.
- Recovery type. This is reserved for future use.
- Recovery level. This is reserved for future use.
- Resource variable name, which is used for Event Manager events.
- Instance vector, which is used for Event Manager events. This is a set of elements of the form "name=value". The values uniquely identify the copy of the resource in the system and, by extension, the copy of the resource variable.
- Predicate, which is used for Event Manager events. This is a relational expression between a resource variable and other elements. When this expression is true, the Event Management subsystem generates an event to notify the Cluster Manager and the appropriate application.
- Rearm predicate, which is used for Event Manager events. This is a predicate used to generate an event that alters the status of the primary

predicate. This predicate is typically the inverse of the primary predicate. It can also be used with the event predicate to establish an upper and a lower boundary for a condition of interest.

Each object requires one line in the event definition, even if the line is not used. If these lines are removed, HACMP ES Cluster Manager cannot parse the event definition properly, and this may cause the system to hang. Any line beginning with "#" is treated as a comment line.

Note: The rules file requires exactly nine lines for each event definition, not counting any comment lines. When adding a user-defined event at the bottom of the rules file, it is important to remove the unnecessary empty line at the end of the file, or the node will hang.

Following is an example of an event definition for `node_up`:

```
##### Beginning of the Event Definition: node_up
#
TE_JOIN_NODE
0
/usr/sbin/cluster/events/node_up.rp
2
0
# 6) Resource variable - only used for event management events

# 7) Instance vector - only used for event management events

# 8) Predicate  - only used for event management events

# 9) Rearm predicate - only used for event management events

###### End of the Event Definition: node_up
```

This example is just one of the event definitions that can be found in the `rules.hacmprd` file. In this example, the recovery program `/usr/sbin/cluster/events/node_up.rp` is invoked when the `node_up` event occurs. Values are specified for the state, recovery type, and recovery level. There are four empty lines for resource variable, instance variable, predicate, and rearm predicate.

You can define other events to react to non-standard HACMP ES events. For example, to define the event that the `/tmp` file system is over 90 per cent full, the `rules.hacmprd` file must be modified.

Many events are predefined in the IBM Parallel System Support Program (PSSP). These events can be exploited (when used within user-defined events) as follows:

1. Stop the cluster.

2. Edit the `rules.hacmprd` file. Back up the file before modifying it. Add the predefined PSSP event manually. If you need synchronizing points across all nodes in the cluster, use the **barrier** command in the recovery program. (Read more about the barrier command, and synchronization of recovery programs in the HACMP Concepts, Installation, and Administration Guides.)
3. Restart the cluster. The `rules.hacmprd` file is stored in memory when Cluster Manager is started. To accurately implement the changes, restart all the clusters. There should not be any inconsistent rules in a cluster.
4. Cluster Manager uses all events in the `rules.hacmprd` file.

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.

Resource states are represented by resource variables. Resource conditions are represented as expressions called predicates.

Event Management receives resource variables from the Resource Monitor, which observes the state of specific system resources and transforms this state into several resource variables. These variables are periodically passed to Event Management. Event Management applies predicates that are specified by the HACMP ES Cluster Manager in `rules.hacmprd` for each resource variable. When the predicate is evaluated as being true, an event is generated and sent to the Cluster Manager. Cluster Manager initiates the voting protocol, and the recovery program file (`xxx.rp`) is run (according to event priority) on a set of nodes specified by ″node sets″ in the recovery program.

The recovery program file (*xxx*`.rp`) is made up of one or more recovery program lines. Each line is declared in the following format:

```
relationship     command_to_run     expected_status     NULL
```

There must be at least one space between each value in the line. ″Relationship″ is a value used to decide which program should run on which kind of node. Three types of relationship are supported:

- All. The specified command or program is run on all nodes of the current HACMP cluster.
- Event. The specified command or program is run only on the nodes on which the event occurred.
- Other. The specified command or program is run on all nodes on which the event did *not* occur.

″Command_to_run″ is a quotation mark-delimited string, with or without a full-path specification to an executable program. Only HACMP-delivered

event scripts can use a relative-path definition. Other scripts or programs must use the full-path specification, even if they are located in the same directory as the HACMP event scripts.

″Expected_states″ is the return code of the specified command or program. It is either an integer value, or an ″x″. If ″x″ is used, Cluster Manager does not care about the return code. All other codes must be equal to the expected return code, otherwise Cluster Manager detects the event failure. The handling of this event ″hangs″ the process until recovery (through manual intervention) occurs. Without manual intervention, the node does not synchronize with the other nodes. Synchronization across all nodes is required for the Cluster Manager to control all the nodes.

″NULL″ is a field reserved for future use. The word ″NULL″ must appear at the end of each line except the barrier line. If you specify multiple recovery commands between two barrier commands, or before the first one, the recovery commands are run in parallel on the node itself, and between the nodes.

The barrier command is used to synchronize all the commands across all the cluster nodes. When a node hits the barrier statement in the recovery program, Cluster Manager initiates the barrier protocol on this node. Since the barrier protocol is a two-phase protocol, all nodes are notified that both phases have completed when all of the nodes have met the barrier in the recovery program, and ″voted″ to approve the protocol.

The process can be summarized as follows:

1. Either Group Services/ES (for predefined events) or Event Management (for user-defined events) notifies HACMP ES Cluster Manager of the event.
2. Cluster Manager reads the `rules.hacmprd` file, and determines the recovery program that is mapped to the event.
3. Cluster Manager runs the recovery program, which consists of a sequence of recovery commands.
4. The recovery program executes the recovery commands, which may be shell scripts or binary commands. (In HACMP for AIX, the recovery commands are the same as the HACMP event scripts.)
5. Cluster Manager receives the return status from the recovery commands. An unexpected status ″hangs″ the cluster until manual intervention (using `smit cm_rec_aids` or the `/usr/sbin/cluster/utilities/clruncmd` command) is carried out.

HACMP ES Script Files

The following sample scripts for failover recovery and user-defined events are included with DB2 UDB EEE. The script files are located in the `$INSTNAME/sqllib/samples/hacmp/es` directory. The scripts will work "as is", or you can customize the recovery action.

- DB2 database partition recovery script `rc.db2pe`. This is the script file used to start and stop the HACMP configuration on a database partition. It also works as an HACMP start and stop script for an NFS server of the DB2 instance owner.
- DB2-specific user-defined events for HACMP ES. Six default events are included: one for process recovery, two for paging space, and three for NFS and automounter recovery.
- DB2 instance NFS file server failover. This script provides failover recovery of the file system server for a DB2 instance to a backup.
- Network failover. The scripts `network_up_complete`, `network_back`, `network_down_complete`, and `network_down` allow SP DB2 database partitions to failover if their SP switch adapter fails.
- Scripts to define monitoring events for the SP GUI Perspectives. Monitoring of failover and user-defined recovery is possible through the Event and Hardware Perspectives. Read the documentation for PSSP Administration to find out more about Perspectives.
- Installation scripts to install and remove core scripts and events on the HACMP ES nodes.
- Script files to create and remove the SP Perspectives problem management (pman) resources for monitoring the HACMP and DB2 configuration.

The recovery scripts must be installed on each node that will run recovery operations. The script files can be centrally installed from the SP control workstation or other designated SP node:

1. Copy the scripts from the `$INSTNAME/sqllib/samples/hacmp/es` directory to one of either the SP control workstation or another SP node that can run the **pcp** and **pexec** commands. These commands are required for the install operation.
2. Customize the `reg.parms.SAMPLE` and `failover.parms.SAMPLE` files for your environment by setting key parameters (such as BUFFPAGE) for failover configurations. Typically, for mutual takeover configurations, your failure settings will be adjusted lower to one-half the size of your regular settings or less. Also, you will use a copy of these files renamed with your own name (instead of "SAMPLE").
3. Customize (as necessary) the five parameters NFS_RETRIES, START_RETRIES, MOUNT_NFS, STOP_RETRIES, and FAILOVER in the `rc.db2pe` file. The retry and failover settings should be adequate for most implementations. The MOUNT_NFS setting should be configured, depending on whether you will be using the package for NFS server

availability. You should specify this setting if you want rc.db2pe to mount and verify the NFS home directory of the DB2 instance owner for you. Setting the FAILOVER parameter to "YES" will invoke db2_proc_restart and launch an attempt to restart a DB2 database partition. If the restart operation is unsuccessful, HACMP will shut down with a failover.

4. Customize db2_paging_action, db2_proc_recovery, and nfs_auto_recovery in the event file. Edit pwq to change this to the DB2 instance owner. Customize db2_paging_action to specify which action is to be taken if paging space becomes more that ninety percent full. (If this does occur, the DB2 database partition is stopped.) Modify the script if additional recovery actions are required.
5. Use db2_inst_ha to install the scripts and events on the nodes you specify. (HACMP ES must be pre-installed on these nodes before you begin.) The syntax of db2_inst_ha is:

   ```
   db2_inst_ha $INSTNAME/sqllib/samples/hacmp/es <nodelist> <DATABASENAME>

   where

      $INSTNAME/sqllib/samples/hacmp/es is the directory in which the
         scripts and the event are located
      <nodelist> is the pcp or pexec style of the nodes; for example,
         1-16 or 1,2,3,4
      <DATABASENAME> is the name of the database for regular and
         failover parameter files.
   ```

   The reg.parms.SAMPLE and failover.parms.SAMPLE files will be copied to each node and renamed reg.parms.DATABASENAME. db2_inst_ha copies files to each node in /usr/bin, and updates the HACMP event files:

   ```
   /usr/sbin/cluster/events/rules.hacmprd
   /usr/sbin/cluster/events/network_up_complete
   /usr/sbin/cluster/events/network_down_complete
   ```

6. Configure your system and scripts with HACMP.
7. Use the **create_db2_events** command to install the monitoring events for problem management resources (pman) and the SP GUI Perspectives. Additional configuration and customization in Perspectives is needed. For more information about Perspectives, read the PSSP Administration Guide.
8. Use the ha_db2stop command to shutdown the database partitions without HACMP ES failover recovery taking place. To use this command, copy the file to the database user's home directory and make sure permissions and ownership are set for that user. To stop the database without failover recovery, then as that user, type:

   ```
   ha_db2stop
   ```

Note: You must wait for the command to return. Exiting by using a `ctrl-C` interrupt, or by killing the process, may re-enable failover recovery prematurely, and some database partitions may not be stopped.

DB2 Recovery Script Operations with HACMP ES

HACMP ES invokes the DB2 recovery scripts in the following way:

- `node_up_local` (starting a node)

  HACMP runs the `node_up` sequence, acquiring volume groups, logical volumes, file systems, and IP addresses specified in resource groups that are owned (through cascading) or assigned (through rotating) to this node.

  When `node_up_local_complete` is run, the application server definition that contains `rc.db2pe` is initiated to start the database partition specified in the application server definitions on this physical node.

  Note: `rc.db2pe`, when running in start mode, adjusts the DB2 parameters specified in `reg.parms.DATABASE` for each DATABASE in the database directory that matches a parameter (parms) file.

  Each node follows this sequence when starting. If you have multiple HACMP clusters and start them in parallel, multiple nodes are brought up at once.

- `node_down_remote` (failover)

  HACMP acquires volume groups, logical volumes, file systems, and IP addresses that are specified in the resource group on the designated takeover node.

  When `node_down_remote_complete` is run, HACMP will run `rc.db2pe` as the application server start script specified in the resource group for this database partition.

  Note: `rc.db2pe`, when running in mutual takeover mode, stops the DB2 database partition running on it, adjusts the DB2 parameters specified in `failover.parms.DATABASE` for each DATABASE in the database directory that matches a parameter (parms) file, and then starts both database partitions on the physical takeover node.

- `node_up_remote` (reintegration of a failed node - cascading mutual takeover resource group)

  When `node_up_remote` is run on the old takeover node, the application server definition causes `rc.db2pe` to be run in stop mode.

  Note: `rc.db2pe`, when running in a reintegration mode (mutual takeover), stops both of the database partitions running on it, adjust the DB2 parameters specified in `reg.parms.DATABASE` for each DATABASE in

the database directory that matches a parameter (parms) file, and then starts just the database partition to be kept on this physical takeover node.

The old takeover node releases volume groups, logical volumes, file systems, and IP addresses specified in resource groups that are to be owned by the reintegrating node.

HACMP re-acquires volume groups, logical volumes, file systems, and IP addresses specified in the resource group that is now owned by the reintegrating node.

When `node_up_local_complete` is run, the application server definition that contains `rc.db2pe` is initiated to start the DB2 database partition specified in the application server definition on this reintegrating physical node.

Note: `rc.db2pe`, when running in start mode, adjusts the DB2 parameters specified in `reg.parms.DATABASE` for each DATABASE in the database directory that matches a parameter (parms) file.

- `node_down_local` (node stop or stop with takeover)

  When `node_down_local` is run on the stopping node, the application server definition causes `rc.db2pe` to be run in stop mode.

  Note: `rc.db2pe`, when running in stop mode, adjusts the DB2 parameters specified in `failover.parms.DATABASE` for each DATABASE in the database directory that matches a parameter (parms) file, and then stops the DB2 database partition (this is for takeover).

  HACMP releases volume groups, logical volumes, file systems, and IP addresses specified in resource groups that are now owned by the node.
- `db2_proc_recovery` (db2 process death)

  All nodes run the `db2_proc_restart` script. The node on which the failure occurred restarts the correct DB2 database partition.
- `db2_paging_recovery` (paging space recovery)

  All nodes run the `db2_paging_action` script. If a node has more than 70 percent of paging space filled, a wall command is issued. If a node has more than 90 percent of paging space filled, DB2 database partitions on this physical node are stopped and then restarted.
- `nfs_auto_recovery` (nfs or automount process failure)

  All nodes run the `rc.db2pe` script in NFS mode. If an NFS process stops running, it is restarted. Similarly, if the automount process stops running, it is restarted.
- `network_down_complete` (network failure - SP switch)

The net_down script is called. This verifies the network as the SP switch network, and verifies that it is down. If that is the case, it waits a user-defined time interval. The default time interval is 100 seconds.

If the SP switch network comes back, as indicated by an network_up_complete event, no recovery is effected.

If the time limit is reached, HACMP is stopped with failover.

Note: All events can be monitored through SP problem management and the SP Perspectives GUI.

Other Script Utilities

Other script utilities are available for your use, including:

- ha_cmd, a command provided to start HACMP on SP nodes from the control workstation. The syntax is:

```
ha_cmd <noderange> <START|STOP|TAKE|FORCE>

where

   <noderange> is a pcp or pexec style of SP noderange.
      For example, "ha_cmd 3-6 START" would start HACMP on nodes 3,4,5,6.
                   "ha_cmd 5 TAKE" would shut down HACMP on node 5
                      for mutual takeover.
```

- ha_mon, a command for monitoring HACMP hacmp_out files from the SP control workstation. The syntax is:

```
ha_mon <node>

where

   <node> is the SP node to be monitored.

   ha_mon will "tail -f" the /tmp/hacmp.out file on the node you specify.
```

- db2_turnoff_recov, a command for temporarily disabling all HACMP (non-failover) recovery, and designed for extremely rare situations. No DB2 process, paging, NFS, or automounter recovery is initiated. This function removes the event stanzas for that recovery from the HACMP rules file. HACMP must be stopped and restarted. The syntax is:

```
db2_turnoff_recov <nodelist>
```

- db2_turnon_recov, a command to re-enable HACMP (non-failover) recovery. This command is used after db2_turnoff_recov to restore HACMP rules files, so that user-defined event recovery can occur. HACMP must be stopped and restarted. The syntax is:

```
db2_turnon_recov <nodelist>
```

Monitoring HACMP Clusters

Scripts are provided for creating SP problem management (pman) events to monitor the DB2 HACMP ES configuration, in addition to those monitoring utilities already present in HACMP ES. To monitor HACMP status from the SP control workstation:

- Install the HACMP client code on the control workstation.
- Edit the `/usr/sbin/cluster/etc/clhosts` file, and include the SP ethernet IP addresses of the nodes that you want to monitor.
- Invoke the command `startsrc -s clinfo` to start monitoring the clusters.

HACMP supplies an interface for monitoring the clusters (`/usr/sbin/cluster/clstat`.

To use the problem management monitoring with SP Perspectives GUI for HACMP RS and user-defined events:

1. Invoke `create_db2_events <nodelist>`, where *nodelist* contains pcp or pexec style nodes. This script creates five pman events for monitoring by Perspectives.

   Note: The resource variables `PSSP.pm.User_state12-16` are used in the creation of these events. If these resource variables are already being used for some other purpose, `create_db2_events` and `update_db2_events` must be updated to use different resource variables.
2. Start Perspectives on the control workstation. From the launch pad, choose the event perspective. You should see five events: `db2_hacmp_recovery`, `db2_process_recovery`, `db2_paging_err`, `db2_nfs_err`, and `Errlog_PERM_entry`.
3. Double-click on each event. On the screen that appears, register (within the Definition Table) a condition for the event. Click next to the down arrow by `Name: "unnamed"`, and select the same name as the event you specify as the condition. Select the `"Response Options"` tab. Click on the button at the top of the display (″Send Message to Perspectives event session″). You can specify commands, errlog entries, as well as SNMP traps for these event occurrences. The event log displays are maintained only across Perspective sessions; therefore, you might want to create AIX error log entries for each. Select **OK**, and close the window.
4. From the Perspectives launch pad, select the hardware Perspective.
5. When the hardware frame GUI appears, select ″View″ and then ″Monitor″. You are provided with a list of events that can be monitored for your SP. Scrolling to the bottom of the list, you will find two additional events: one for HACMP DB2 recovery (`db2_ha_ind`), and the other for SP node PERM errors (`Errlog_PERM_mon`. Select those that you want to monitor. (When an

event occurs, the node displays a red "X". If all monitored conditions are fine, the node display is green.) `host_responds`, `switch_responds`, and `node_power_LED` are typically used. You can also monitor the DB2 HACMP recovery, as well as PERM errors, on the node.

Note: The `db2_hacmp_mon` and `db2_hacmp_recovery` variables for pman and Perspectives do not reflect HACMP cluster status. Rather, these variables reflect the status of the `rc.db2pe` operation to start or stop DB2. The "real" HACMP status is shown in the HACMP `clstat` monitor, and reflects the HACMP cluster state. If you want `db2_hacmp_ind` to reflect monitoring similar to HACMP status, add the following line to your `/etc/inittab` file:

```
haind:2:wait:/usr/bin/db2_update_events HAIND OFF 2>&1 >/dev/null
```

If you are planning to use NetView for your implementation, consider using HAVIEW (which is part of HACMP) for monitoring your configuration. For information about configuring that product, refer to the NetView documentation.

DB2 SP HACMP ES Installation

To help you plan for the installation of HACMP ES with DB2 Universal Database, following is a step-by-step overview of the installation and migration processes.

DB2 SP HACMP ES New Installation

To install HACMP ES:

1. Install the AIX operating system on each SP node, (refer to the SP Installation and Administration Guides). Ensure that proper paging space is available on both the control workstation, and each of the SP nodes. Ensure that switch configuration has been considered and implemented, along with any other modifiable configuration parameters. Put in place the SP monitoring (Perspectives) that you want to use. Ensure that the SP dsh, pcp, and pexec commands work.
2. Design your database layout. This should, at a minimum, include the number of nodes to be used, the mapping of DB2 database partitions to physical nodes, the disk requirements per node or partition, and table space considerations. You should also consider who the main DB2 instance owner will be, and what access authorization this and other users will require.
3. Plan your external SSA disk configuration, including redundant adapters, mirrored disks, and the twin-tailing of disks.
4. Using your database layout and SSA configuration, complete the HACMP worksheets located in the HACMP Planning, Installation, and Administration Guides.

5. Implement your external SSA disk configuration. Ensure that microcode levels are consistent across all drives, and use the Maymap utility to validate and fill in any gaps in your worksheets.
6. Install DB2 UDB EEE on each SP node.
7. Install HACMP ES on each SP node.
8. Install the DB2 UDB EEE HACMP ES on SP Package, using the **db2_inst_ha** command.
9. Create the main DB2 instance user, and ensure that it can access all nodes. This is not a highly available user at this point. This can be temporarily an SP user on the SP control workstation.
10. Create your DB2 instance and database. Ensure that it is operational by invoking **db2start**, and then **db2stop**, before proceeding to the next step.
11. If you want to load the database before adding HACMP, do this now.
12. Configure HACMP ES on the SP nodes topology and resource groups according to the HACMP worksheets and the information in this document.
13. Beginning with your NFS server node for the main DB2 instance user, change this user (by modifying `/etc/security/user` and `/etc/passwd`) on all nodes, in accordance with what is specified in this document. This user will become a highly available NFS user; and this node and its backup will update `/etc/exports`. All nodes will be able to mount this directory using NFS (with an entry in `/etc/filesystems` on each node) through the switch alias IP addresses.
14. ″Tar″ the home directory of the main instance user and ″un-tar″ the home directory in the new location.
15. Create an NFS file system on each of the SP nodes to mount a new main instance home directory.
16. Start HACMP on the NFS server node. Verify that it comes up successfully by investigating `/tmp/hacmp.out`. The **ha_mon** command can be used to monitor this file as it is written.
17. Bring up the other nodes one at a time, verifying each successful completion by investigating `/tmp/hacmp.out`. The **ha_mon** command can be used to monitor this file as it is written.
18. Set up the optional monitoring through Perspectives and Problem Management.
19. Validate failover functionality on each node by simulating a concurrent maintenance action on each node. The **ha_cmd** command (specifying the TAKE option) can be used to stop HACMP gracefully with takeover. Verify that the takeovers and the reintegrations are successful by interrogating `/tmp/hacmp.out` and using your monitoring tools.

DB2 SP HACMP ES Migration

If you are migrating from a non-HACMP installation to one with HACMP, consider the following overview:

1. Convert your existing external disks to a highly available, twin-tailed, mirrored configuration. Add any extra hardware and disks to achieve this configuration, remembering that names of different logical volumes on different nodes *must* be unique when they are twin-tailed. This applies to volume groups, logical volumes, and file systems.
2. Complete the HACMP planning and the related worksheets, including the worksheets in this document.
3. Implement your external SSA disk configuration changes. Ensure that microcode levels are consistent across all drives, and use the Maymap utility to validate and eliminate any gaps in the worksheets.

   Note: SSA disks in a RAID5 configuration is supported. Two SSA adapters in the same RAID loop is the only configuration permitted. For an HACMP configuration with the RAID disks twin-tailed, only one adapter per node is supported. In this configuration, the adapter is a single point of failure for access to the disks, and extra configuration is recommended to detect the adapter outage and promote this to an HACMP failover event. AIX error notification is the simplest way to configure a node for failover, should the SSA adapter fail. Refer to *HACMP for AIX, V4.2.2, Enhanced Scalability Installation and Administration Guide* for more information about AIX error notification.
4. Install HACMP ES on each SP node.
5. Install the DB2 UDB EEE HACMP ES on SP Package, using the **db2_inst_ha** command.
6. Configure HACMP ES on the SP nodes topology and resource groups according to the HACMP worksheets and the information in this document.
7. Beginning with your NFS server node for the main DB2 instance user, change this user (by modifying `/etc/security/user` and `/etc/passwd` on all nodes, in accordance with what is specified in this document. This user will become a highly available NFS user; and this node and its backup will update `/etc/exports`. All nodes will be able to mount this directory using NFS (with an entry in `/etc/filesystems` on each node) through the switch alias IP addresses.
8. ″Tar″ the home directory of the main instance user and ″un-tar″ the home directory in the new location.
9. Create an NFS file system on each of the SP nodes to mount a new main instance home directory.

10. Start HACMP on the NFS server node. Verify that it comes up successfully by investigating `/tmp/hacmp.out`. The **ha_mon** command can be used to monitor this file as it is written.
11. Bring up the other nodes one at a time, verifying each successful completion by investigating `/tmp/hacmp.out`. The **ha_mon** command can be used to monitor this file as it is written.
12. Set up the optional monitoring through Perspectives and Problem Management.
13. Validate failover functionality on each node by simulating a concurrent maintenance action on each node. The **ha_cmd** command (specifying the TAKE option) can be used to stop HACMP gracefully with takeover. Verify that the takeovers and the reintegrations are successful by interrogating `/tmp/hacmp.out` and using your monitoring tools.

DB2 SP HACMP ES Worksheets

The following worksheets are designed to be used with HACMP worksheets that should be completed in preparation for your external SSA disk configuration (and that are located in the HACMP Planning, Installation, and Administration Guides). In each case, both a completed example, and a blank worksheet, are provided.

The database configuration on external disks documented in the first sample worksheet is shown in the following figure. The statement used to create the database is:

```
db2 create database pwq on /newdata
```

Both SSA external adapters and external SSA disks are mirrored and twin-tailed for logical volumes with no single point of failure. The diagram depicts a configuration that is similar to output from the **maymap** command. Maymap is a utility (available through AIXTOOLS) that shows the external SSA disk configuration, and should be used when planning your setup.

Sample DB2 4-node Database External Disks Setup

- Showing twin-tailing for High Availability.

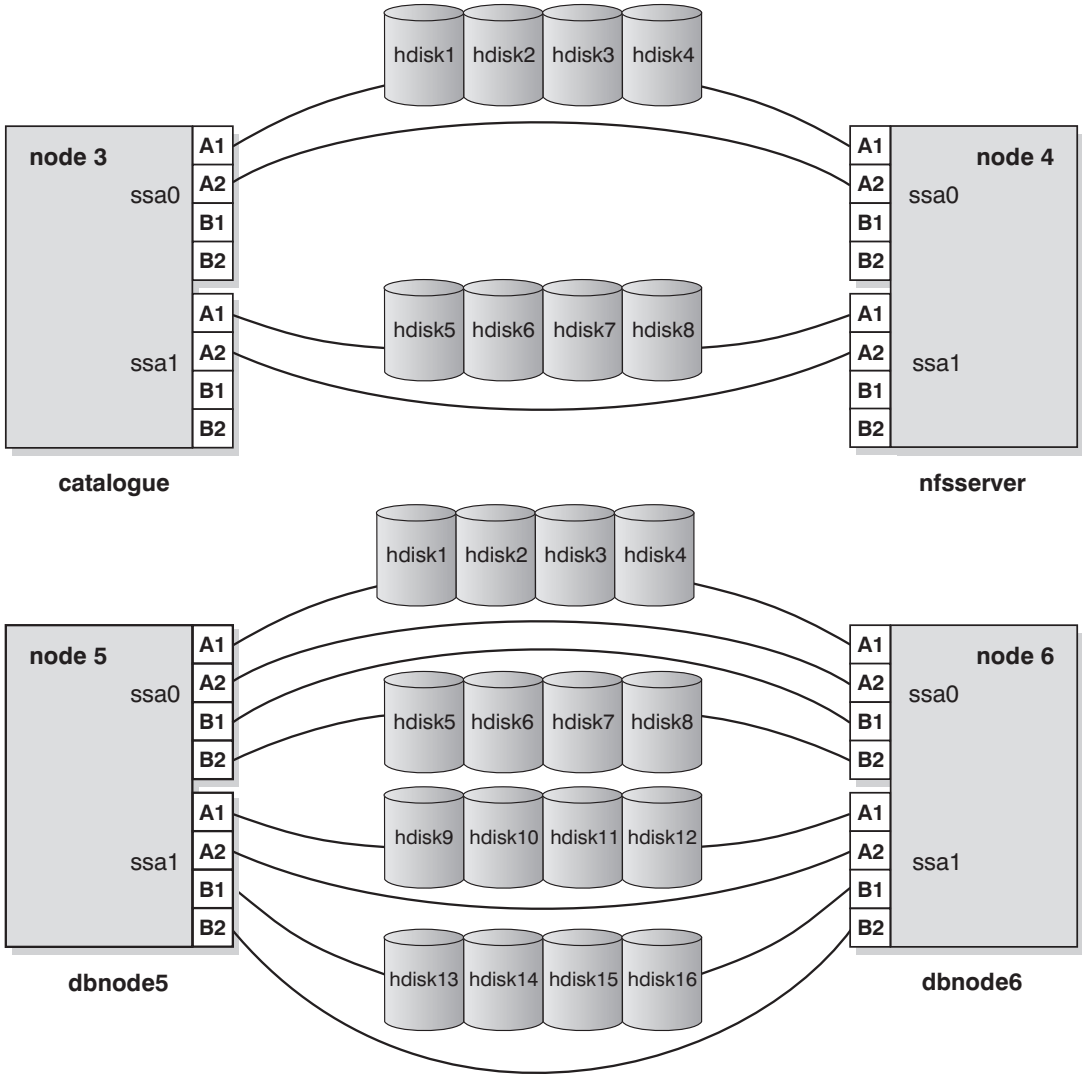


Figure 27. Sample DB2 4-node Database External Disks Setup

Before you review the following table, you should read the HACMP documentation regarding the quorum settings on volume groups, and mirrored write consistency settings on logical volumes. The settings used for both will directly affect your availability and performance. Ensure that you

review these settings and understand their implications. The typical setting for both ″quorum″ and ″mirrored write consistency″ is ″off″.

Table 12. HACMP Volume Groups, Logical Volumes, and File Systems

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Copies	hdisk List	File System Mount Point	File System Log Logical Volume	Node Description and Backup	User Owner of /dev Logical Device
3	havg3	8	hlv300	10	2	hdisk1 hdisk5	/newdata /pwq /NODE0003	hlog301	Catalognode mount point; node 4	root *
3	havg3	8	hlog301	1	2	hdisk1 hdisk5	N/A	N/A	Catalognode jfslog; node 4	root *
3	havg3	8	hlv301	10	2	hdisk2 hdisk6	N/A	N/A	Catalognode rawtemp space; node 4	pwq **
4	havg4	8	hlv400	10	2	hdisk3 hdisk7	/dbmnt	hlog401	nfsserver pwq home; node 3	root *
4	havg4	8	hlog401	1	2	hdisk3 hdisk7	N/A	N/A	nfsserver jfslog; node 3	root *
5	havg5	8	hlv500	10	2	hdisk1 hdisk9	/newdata/ pwq/ NODE0005	HLOG501	Dbnode5 mount point; node 6	root *
5	havg5	8	hlog501	1	2	hdisk1 hdisk9	N/A	N/A	Dbnode5 jfslog; node 6	root *
5	havg5	8	hlv501	10	2	hdisk2 hdisk10	N/A	N/A	Dbnode5 raw temp space; node 6	pwq **
5	havg5	8	hlv502	100	2	hdisk2 hdisk10	N/A	N/A	Dbnode5 raw table space; node 6	pwq **
5	havg5	8	halv503	100	2	hdisk3 hdisk11	N/A	N/A	Dbnode5 raw table space; node 6	pwq **

Table 12. HACMP Volume Groups, Logical Volumes, and File Systems (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Copies	hdisk List	File System Mount Point	File System Log Logical Volume	Node Description and Backup	User Owner of /dev Logical Device
5	havg5	8	halv504	100	2	hdisk3 hdisk11	N/A	N/A	Dbnode5 raw table space; node 6	pwq **
5	havg5	8	halv505	100	2	hdisk4 hdisk12	/dbdata5	hlog501	Dbnode6 system table space; node 6	root *
6	havg6	8	hlv600	10	2	hdisk5 hdisk13	/newdata/ pwq/ NODE0006	hlog601	Dbnode6 mount point; node 5	root *
6	havg6	8	hlog601	1	2	hdisk5 hdisk13	N/A	N/A	Dbnode6 jfslog; node 5	root *
6	havg6	8	hlv601	10	2	hdisk6 hdisk14	N/A	N/A	Dbnode6 raw temp space; node 5	pwq **
6	havg6	8	hlv602	100	2	hdisk6 hdisk14	N/A	N/A	Dbnode6 raw table space; node 5	pwq **
6	havg6	8	hlv603	100	2	hdisk7 hdisk15	N/A	N/A	Dbnode6 raw table space; node 5	pwq **
6	havg6	8	hlv604	100	2	hdisk7 hdisk15	N/A	N/A	Dbnode6 raw table space; node 5	pwq **
6	havg6	8	hlv605	100	2	hdisk8 hdisk16	/dbdata6	hlog601	Dbnode6 system table space; node 5	root *

Table 12. HACMP Volume Groups, Logical Volumes, and File Systems (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Copies	hdisk List	File System Mount Point	File System Log Logical Volume	Node Description and Backup	User Owner of /dev Logical Device
Notes: 1. * jfs file system logical volumes and logs keep root permissions. 2. ** raw database spaces get database user permissions on /dev raw file entries (/dev/rxxxx).										

Table 13. HACMP Volume Groups, Logical Volumes, and File Systems - Blank

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Copies	hdisk List	File System Mount Point	File System Log Logical Volume	Node Description and Backup	User Owner of /dev Logical Device

Table 13. HACMP Volume Groups, Logical Volumes, and File Systems - Blank (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Copies	hdisk List	File System Mount Point	File System Log Logical Volume	Node Description and Backup	User Owner of /dev Logical Device

Table 13. HACMP Volume Groups, Logical Volumes, and File Systems - Blank (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Copies	hdisk List	File System Mount Point	File System Log Logical Volume	Node Description and Backup	User Owner of /dev Logical Device

Table 14. Planning HACMP NFS Server

SP Node	External File System	Backup Node	SP Switch Boot and Service IP Alias Pairs	File System to Mount (/etc/filesystems)	File System to Specify as Database Home Directory	Addresses to which File System is to be Exported (/etc/exports)
3	/dbmnt	4	nfs_boot_3 nfs_client_3	nfs_server:/ dbmnt as /dbi	/dbi/pwq	nfs_boot_3 nfs_client_3 nfs_server_boot nfs_server nfs_boot_5 nfs_client_5 nfs_boot_6 nfs_client_6

Table 14. Planning HACMP NFS Server (continued)

SP Node	External File System	Backup Node	SP Switch Boot and Service IP Alias Pairs	File System to Mount (/etc/filesystems)	File System to Specify as Database Home Directory	Addresses to which File System is to be Exported (/etc/exports)
4	/dbmnt	3	nfs_server_boot nfs_server	nfs_server:/ dbmnt as /dbi	/dbi/pwq	nfs_boot_3 nfs_client_3 nfs_server_boot nfs_server nfs_boot_5 nfs_client_5 nfs_boot_6 nfs_client_6
5	N/A	N/A	nfs_boot_5 nfs_client_5	nfs_server:/ dbmnt as /dbi	/dbi/pwq	N/A
6	N/A	N/A	nfs_boot_6 nfs_client_6	nfs_server:/ dbmnt as /dbi	/dbi/pwq	N/A

Notes:

1. `/etc/passwd` must be the same on all nodes. This can be synchronized from the control workstation.
2. Ensure that the external file system has the permission of the database instance owner.
3. The `/etc/filesystems` must have the mount parameters: hard, bg, intr, and rw.
4. The `/etc/exports` will have

   ```
   -root=ip1:ip2:ip3
   ```

   only on the server and its backup.

Table 15. Planning HACMP NFS Server - Blank

SP Node	External File System	Backup Node	SP Switch Boot and Service IP Alias Pairs	File System to Mount (/etc/filesystems)	File System to Specify as Database Home Directory	Addresses to which File System is to be Exported (/etc/exports)

Table 15. Planning HACMP NFS Server - Blank (continued)

SP Node	External File System	Backup Node	SP Switch Boot and Service IP Alias Pairs	File System to Mount (/etc/filesystems)	File System to Specify as Database Home Directory	Addresses to which File System is to be Exported (/etc/exports)

Table 15. Planning HACMP NFS Server - Blank (continued)

SP Node	External File System	Backup Node	SP Switch Boot and Service IP Alias Pairs	File System to Mount (/etc/filesystems)	File System to Specify as Database Home Directory	Addresses to which File System is to be Exported (/etc/exports)

Chapter 7. High Availability on the Windows Operating System

You can set up your database system so that if a machine fails, the database server on the failed machine can run on another machine. On Windows NT, failover support can be implemented with Microsoft Cluster Server (MSCS). To use MSCS, you require Windows NT Version 4.0 Enterprise Edition with the MSCS feature installed.

MSCS can perform both failure detection and the restarting of resources in a clustered environment, such as failover support for physical disks and IP addresses. (When the failed machine is online again, resources will not automatically fall back to it, unless you previously configure them to do so. For more information, see "Fallback Considerations" on page 233.)

Before you enable DB2 instances for failover support, perform the following planning steps:

1. Decide which disks you want to use for data storage. Each database server should be assigned at least one disk for its own use. The disk that you use to store data must be attached to a shared disk subsystem, and must be configured as an MSCS disk resource.
2. Ensure that you have one IP address for each database server that you want to use to support remote requests.

When you set up failover support, it can be for an existing instance, or you can create a new instance when you implement the failover support.

To enable failover support, perform the following steps:

1. Create an input file for the DB2MSCS utility.
2. Invoke the **db2mscs** command.
3. If you are using a partitioned database system, register database drive mapping to enable mutual takeover. See "Registering a Database Drive Mapping for Mutual Takeover Configurations in a Partitioned Database Environment" on page 233.

After you finish enabling the instance for failover support, your configuration will resemble Figure 28 on page 222.

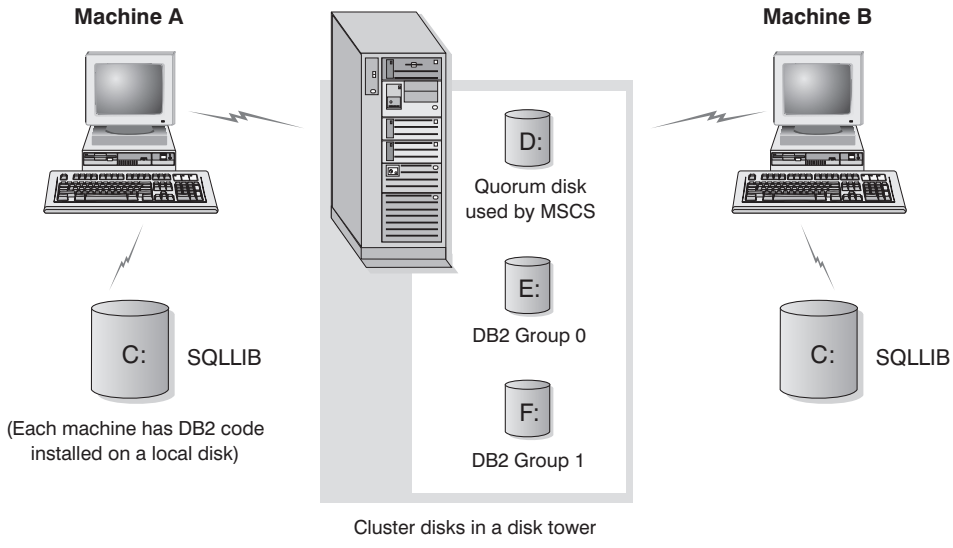


Figure 28. Example MSCS Configuration

The following sections describe the different types of failover support, and how to implement them. Before performing any of the steps described below, you must already have the MSCS software installed on every machine that you want to use in an MSCS cluster. In addition, you must also have DB2 installed on every machine.

Failover Configurations

Two types of configuration are available:

- Hot standby
- Mutual takeover

Currently, MSCS supports clusters of two machines.

In a partitioned database environment, the clusters do not all have to have the same type of configuration. You can have some clusters that are set up to use hot standby, and others that are set up for mutual takeover. For example, if your DB2 instance consists of five workstations, you can have two machines set up to use a mutual takeover configuration, two to use a hot standby configuration, and one machine not configured for failover support.

Hot Standby Configuration

In a hot standby configuration, one machine in the MSCS cluster provides dedicated failover support, and the other machine participates in the database system. If the machine participating in the database system fails, the database server on it will be started on the failover machine. If, in a partitioned

database system, you are running multiple logical nodes on a machine and it fails, the logical nodes will be started on the failover machine. Figure 29 shows an example of a hot standby configuration.

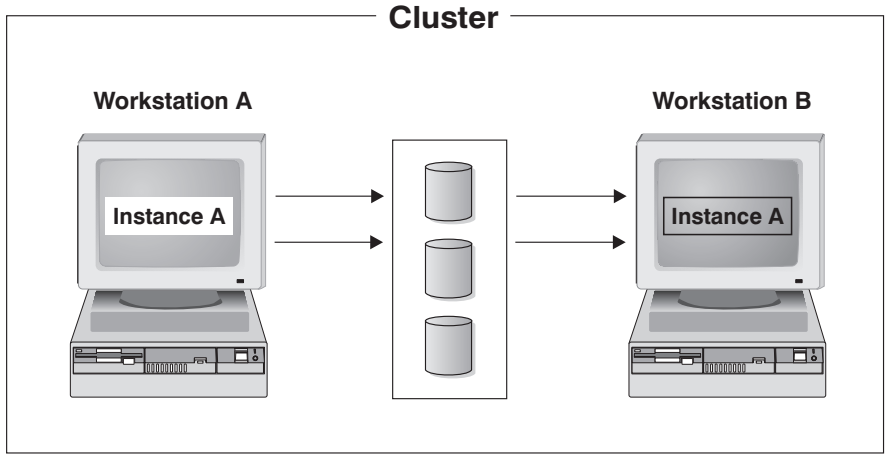


Figure 29. Hot Standby Configuration

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 30 on page 224 shows an example of a mutual takeover configuration.

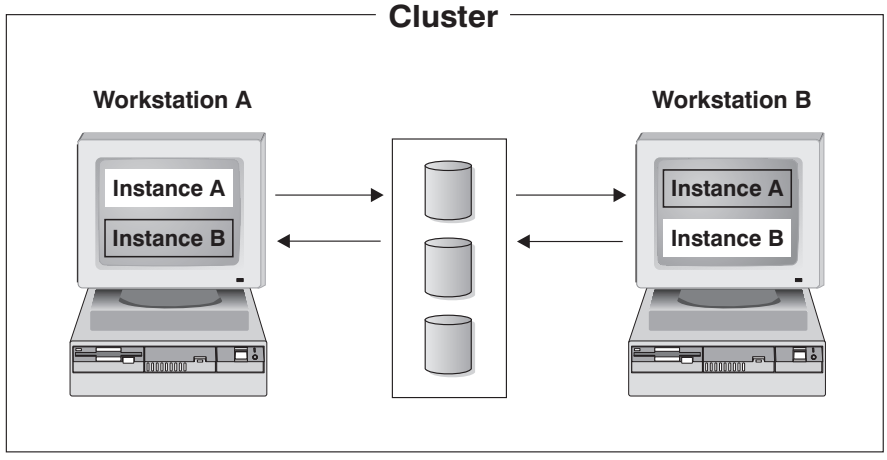


Figure 30. Mutual Takeover Configuration

Using the DB2MSCS Utility

Use the DB2MSCS utility to create the infrastructure for DB2 to support failover in the Windows NT environment using Microsoft Cluster Service (MSCS) support. You can use this utility to enable failover in both single-partition and partitioned database environments.

For the DB2MSCS utility to run successfully, the Cluster Service must be able to locate the resource DLL, `db2wolf.dll`, which resides under the `%ProgramFiles%\SQLLIB\bin` directory. The DB2 UDB Version 7 Installation Program sets the PATH system environment variable to point to the `%ProgramFiles%\SQLLIB\bin` directory. However, it is not required that you reboot the machine after installation if you are running on the Windows 2000 operating system. If you want to run the DB2MSCS utility, you must reboot the machine so that the PATH environment variable is updated for the Cluster Service.

Invoke the **db2mscs** command once for each instance on its instance-owning machine. If there is only one DB2 instance running on one machine in the MSCS cluster, this sets up a hot-standby configuration. If you have an instance running on each machine in the MSCS cluster, you would run DB2MSCS once on each instance-owing machine to set up a mutual takeover configuration.

The DB2MSCS utility:

1. Reads the required MSCS and DB2 parameters from an input file called `DB2MSCS.CFG`. See "Specifying the DB2MSCS.CFG File" on page 225 for information about the full set of input parameters.
2. Validates the parameters in the input file.

3. Registers the DB2 resource type.
4. Creates the MSCS group (or groups) to contain the MSCS and DB2 resources.
5. Creates the IP resource.
6. Creates the Network Name resource.
7. Moves MSCS disks to the group.
8. Creates the DB2 resource (or resources).
9. Adds all required dependencies for the DB2 resource.
10. Converts the non-clustered DB2 instance into a clustered instance.
11. Brings all resources online.

The command syntax is as follows:

Where:

-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.

Specifying the DB2MSCS.CFG File

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=WOLFPACK
GROUP_NAME=DB2 Group
IP_ADDRESS=9.21.22.89
```

Two example configuration files are in the /CFG subdirectory of the /SQLLIB 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. If the instance name is *not* specified, the default instance (the value of the DB2INSTANCE environment variable) is used.

This parameter has a global scope, and you specify it only once in the DB2MSCS.CFG file.

This parameter is optional.

Example:

```
DB2_INSTANCE=DB2
```

The instance must already exist. For information about creating instances, refer to the *DB2 Enterprise - Extended Edition for Windows Quick Beginnings* book.

DB2_LOGON_USERNAME

The name of the logon account for the DB2 service.

This parameter has a global scope, and you specify it only once in the DB2MSCS.CFG file.

This parameter is only required for DB2 Enterprise - Extended Edition instances.

Example:

```
DB2_LOGON_USERNAME=db2user
```

DB2_LOGON_PASSWORD

The password of the logon account for the DB2 service. If the DB2_LOGON_USERNAME parameter is provided, but the DB2_LOGON_PASSWORD parameter is not, the DB2MSCS utility prompts for the password. The password is not displayed when it is typed at the command line.

This parameter has a global scope, and you specify it only once in the DB2MSCS.CFG file.

This parameter is only required for DB2 Enterprise - Extended Edition instances.

Example:

```
DB2_LOGON_PASSWORD=xxxxxx
```

CLUSTER_NAME

The name of the MSCS cluster. All the resources specified following this line are created in this cluster until another CLUSTER_NAME tag is specified.

Specify this parameter once for each cluster.

This parameter is optional. If not specified, the name of the MSCS cluster on the local machine is used.

Example:

```
CLUSTER_NAME=WOLFPACK
```

GROUP_NAME

The name of the MSCS group. If this parameter is specified, a new

MSCS group is created if one does not exist. If the group already exists, it is used as the target group. Any MSCS resource created following this line is created in this group until another GROUP_NAME keyword is specified.

Specify this parameter once for each group.

This parameter is required.

Example:

```
GROUP_NAME=DB2 Group
```

DB2_NODE

The node number of the database partition server (node) to be included in the current MSCS group. If multiple logical nodes exist on the same machine, each node requires a separate DB2_NODE keyword.

You specify this parameter after the GROUP_NAME parameter, so that the DB2 resources are created in the correct MSCS group.

This parameter is only required for DB2 Enterprise - Extended Edition instances.

Example:

```
DB2_NODE=0
```

IP_NAME

The name of the IP Address resource. The value for IP_NAME is arbitrary, but must be unique. When this parameter is specified, an MSCS resource of type IP Address is created.

This parameter is required for remote TCP/IP connections. You must specify this parameter for the instance-owning machine in a partitioned database environment. This parameter is optional in single-partition database environments.

Example:

```
IP_NAME=IP Address for DB2
```

Note: DB2 clients should use the TCP/IP address of this IP resource to catalog the TCP/IP node entry. By using the MSCS IP address, when the database server fails over to the other machine, DB2 clients can still connect to the database server, because the IP address is available on the failover machine.

The attributes of the IP resource are as follows:

IP_ADDRESS

The TCP/IP address of the IP resource. Specify this keyword to set the TCP/IP address for the preceding IP resource.

This parameter is required if the IP_NAME parameter is specified.

Example:

```
IP_ADDRESS=9.21.22.34
```

IP_SUBNET
The subnet mask for the preceding IP resource.

This parameter is required if the IP_NAME parameter is specified.

Example:

```
IP_SUBNET=255.255.255.0
```

IP_NETWORK
The name of the MSCS network to which the preceding IP resource belongs. If this parameter is not specified, the first MSCS network detected by the system is used.

This parameter is optional.

Example:

```
IP_NETWORK=Token Ring
```

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 environments. It is required for partitioned database environments.

Example:

```
NETNAME_NAME=Network name for DB2
```

The attributes of the Network Name resource are as follows:

NETNAME_VALUE
The value for the Network Name.

This parameter is required if the NETNAME_NAME parameter is specified.

Example:

```
NETNAME_VALUE=DB2SRV
```

NETNAME_DEPENDENCY
The dependency list for the Network Name resource. Each Network Name resource must have a dependency on an IP Address resource. If this parameter is not specified, the Network Name resource has a dependency on the first IP resource in the group.

This parameter is optional.

Example:

```
NETNAME_DEPENDENCY=IP Address for DB2
```

DISK_NAME

The name of the physical disk resources to be moved to the current groups. Specify as many disk resources as you need.

Notes:

1. The disk resources must already exist.
2. When the DB2MSCS utility configures the DB2 instance for MSCS 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.

Example:

```
DISK_NAME=Disk E:
DISK_NAME=Disk F:
```

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* MSCS disk that belongs to the same group as the instance directory.

The DB2 instance directory is created on the MSCS disk under the `X:\DB2PROFS` directory (where X is the MSCS disk drive letter).

Example:

```
INSTPROF_DISK=Disk E:
```

TARGET_DRVMAP_DISK

An optional parameter to specify the target MSCS disk for database drive mapping. If a database is created on an MSCS disk that does not belong to the same group as the node, the target drive map disk is used to contain the database partition. If this parameter is not specified, the database drive mapping must be manually registered using the DB2DRVMP utility.

Example:

```
TARGET_DRVMAP_DISK = Disk E:
```

Setting up Failover for a Single-Partition Database System

When you run the DB2MSCS utility against a single-partition database system, one MSCS group contains DB2 and all the dependent MSCS resources (the IP address, Network Name, and disks). For example, the contents of the `DB2MSCS.CFG` file for a single-partition database system will look like the following:

```
#
#  DB2MSCS.CFG for a single-partition database system
#
DB2_INSTANCE=DB2
CLUSTER_NAME=MSCS
GROUP_NAME=DB2 Group
IP_NAME=...
IP_ADDRESS=...
IP_SUBNET=...
IP_NETWORK=...
NETNAME_NAME=...
NETNAME_VALUE=...
DISK_NAME=Disk E:
```

Setting up a Mutual Takeover Configuration for Two Single-Partition Database Systems

You can set up two single-partition database systems, each on a separate machine, so that if the database system on any one machine fails, it is restarted on the other MSCS node.

To set up failover support for this configuration, you need to run the DB2MSCS utility once on each instance-owning machine. You must tailor the configuration file for each database system.

Assume that the DB2 instances are called DB2A and DB2B. The DB2MSCS.CFG file for the DB2A instance would be as follows:

```
#
#  DB2MSCS.CFG for first single-partition database system
#
DB2_INSTANCE=DB2A
CLUSTER_NAME=MSCS
GROUP_NAME=DB2A Group
IP_NAME=...
IP_ADDRESS=...
IP_SUBNET=...
IP_NETWORK=...
NETNAME_NAME=...
NETNAME_VALUE=...
DISK_NAME=Disk E:
```

The DB2MSCS.CFG file for the DB2B instance would be as follows:

```
#
#  DB2MSCS.CFG for second single-partition database system
#
DB2_INSTANCE=DB2B
CLUSTER_NAME=MSCS
GROUP_NAME=DB2B Group
IP_NAME=...
IP_ADDRESS=...
IP_SUBNET=...
```

```
IP_NETWORK=...
NETNAME_NAME=...
NETNAME_VALUE=...
DISK_NAME=Disk F:
```

For a full example, see "Example - Setting up Two Single-Partition Instances for Mutual Takeover" on page 236.

Setting up Multiple MSCS Clusters for a Partitioned Database System

When you run the DB2MSCS utility against a multi-partition database system, one MSCS group is created for each physical machine that participates in the system. The DB2MSCS.CFG file must contain multiple sections, and each section must have a different value for the GROUP_NAME parameter, and for all the required dependent resources for that group.

In addition, you must specify the DB2_NODE parameter for each database partition server in each MSCS group. If you have multiple logical nodes, each logical node requires a separate DB2_NODE keyword.

For example, assume that you have a multi-partition database system that consists of four database partition servers on four machines, and you want to configure two MSCS clusters using mutual takeover configuration. You would set up the DB2MSCS.CFG configuration file as follows:

```
#
#  DB2MSCS.CFG for one partitioned database system with
#  multiple clusters
DB2_INSTANCE=DB2MPP
DB2_LOGON_USERNAME=db2user
DB2_LOGON_PASSWORD=xxxxxx
CLUSTER_NAME=MSCS1
# Group 1
GROUP_NAME=DB2 Group 1
DB2_NODE=0
IP_NAME=...

...
# Group 2
GROUP_NAME=DB2 Group 2
DB2_NODE=1
IP_NAME=...

...

CLUSTER_NAME=MSCS2
# Group 3
GROUP_NAME=DB2 Group 3
DB2_NODE=2
IP_NAME=...

...
# Group 4
```

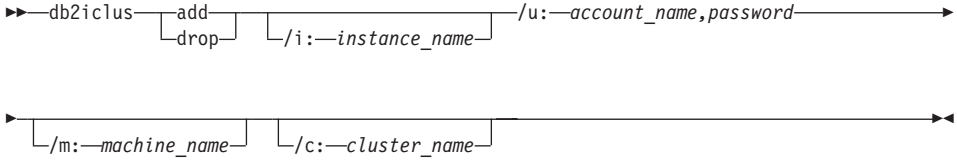
```
GROUP_NAME=DB2 Group 4
DB2_NODE=3
IP_NAME=...

...
```

For a full example, see "Example - Setting up a Four-Node Partitioned Database System for Mutual Takeover" on page 238.

Maintaining the MSCS System

When you run the DB2MSCS utility, it creates the infrastructure for failover support for all machines in the MSCS cluster. To remove support from a machine, use the **db2iclus** command with the "drop" option. To re-enable support for a machine, use the "add" option.

The command syntax is as follows:



Where:

add — Enables failover support on the machine by adding it to an MSCS cluster. The DB2 resource (database server) can then fail over to this machine.

drop — Removes failover support from the machine by dropping it from an MSCS cluster.

/i: *instance_name* — The name of the instance. (This parameter overrides the setting of the DB2INSTANCE environment variable.)

/u: *account_name*, *password* — The domain account used as the logon account name of the DB2 Service. For example:

```
/u:domainA\db2nt,password
```

This parameter is only required with the "add" option.

/m:*machine_name* — The computer name of the machine that you want to add to, or drop from, an MSCS

cluster. You must specify this option if you run the command from a machine other than the one for which you are modifying failover support.

/c: *cluster_name* — The name of the MSCS cluster as it is known on the LAN. This name is specified when the MSCS cluster is first created.

Fallback Considerations

By default, groups are set not to fall back to the original (failed) machine. Unless you manually configure a DB2 group to fall back after failing over, it continues to run on the alternative MSCS node after the cause of the failover has been resolved.

If you configure a DB2 group to automatically fall back to the original machine, all the resources in the DB2 group including the DB2 resource will fall back as soon as the original machine is available. If, during the fall back, a database connection exists, the DB2 resource cannot be brought offline, and the fallback processing will fail.

If you want to force all database connections off the database during fallback processing, set the DB2_FALLBACK registry variable to ON. This variable must be set as follows:

```
db2set DB2_FALLBACK=ON
```

You do not have to reboot or restart the cluster service after setting this registry variable.

Registering a Database Drive Mapping for Mutual Takeover Configurations in a Partitioned Database Environment

When you create a database in a partitioned database environment, you can specify a drive letter to indicate where the database is to be created.

Note: You do not set database drive mapping for single-partition database environments.

When the CREATE DATABASE command runs, it expects that the drive that you specify will be simultaneously available to all of the machines that participate in the instance. Because this is not possible, DB2 uses database drive mapping to assign the same drive a different name for each machine.

For example, assume that a DB2 instance called DB2 contains two database partition servers:

```
NODE0 is active on machine WOLF_NODE_0
NODE1 is active on machine WOLF_NODE_1
```

Assume also that the share disk E: belongs to the same group as NODE0, and that the share disk F: belongs to the same group as NODE1.

To create a database on the share disk E:

```
db2 create database mppdb on e:
```

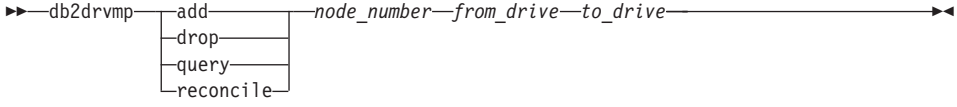
For the command to be successful, drive E: must be available to both machines. In a mutual takeover configuration, each database partition server may be active on a different machine, and the cluster disk E: is only available to one machine. In this situation, the CREATE DATABASE command will always fail.

To resolve this problem, the database drive should be mapped as follows:

```
For NODE0, the mapping is from drive F: to drive E:
For NODE1, the mapping is from drive E: to drive F:
```

Any database access for NODE0 to drive F: is then mapped to drive E:, and any database access for NODE1 to drive E: is mapped to drive F:. Using drive mapping, the CREATE DATABASE command will create database files on drive E: for NODE0 and drive F: for NODE1.

Use the **db2drvmp** command to set up the drive mapping. The command syntax is as follows:



The parameters are as follows:

add Assigns a new database drive map.

drop Removes an existing database drive map.

query Queries a database map.

reconcile Repairs a database map drive when the registry contents are damaged. See “Reconciling the Database Drive Mapping” on page 235 for more information.

node_number The node number. This parameter is required for add and drop operations.

from_drive The drive letter from which to map. This parameter is required for add and drop operations.

to_drive The drive letter to which to map. This parameter is required for add operations. It is not applicable to other operations.

If you wanted to set up database drive mapping from F: to E: for NODE0, you would use the following command:

```
db2drvmp add 0 F E
```

Note: Database drive mapping does not apply to table spaces, containers, or any other database storage objects.

Similarly, to set up database drive mapping from E: to F: for NODE1, you would issue the following command:

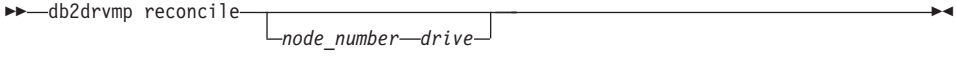
```
db2drvmp add 1 E F
```

Note: Any setup of, or changes to, the database drive mapping does not take effect immediately. To activate the database drive mapping, use the Cluster Administrator tool to bring the DB2 resource offline, then online.

Using the TARGET_DRVMAP_DISK keyword in the `DB2MSCS.CFG` file will enable the drive mapping to be done automatically.

Reconciling the Database Drive Mapping

When a database is created on a machine that has database drive mapping in effect, the map is saved on the drive in a hidden file. This is to prevent the database drive from being removed after the database is created. You will want to *reconcile* the database drive mapping if, for example, you accidentally drop the database drive map. To reconcile the map, run the **db2drvmp reconcile** command for each database partition server that contains the database. The command syntax is as follows:



The parameters are as follows:

node_number The node number of the node to be repaired. If *node_number* is not specified, the command reconciles the mapping for all nodes.

drive The drive to reconcile. If a drive is not specified, the command reconciles the mapping for all drives.

Registering a Database Drive Mapping

The **db2drvmp** command scans all drives on the machine for database partitions that are managed by the database partition server, and reapplies the database drive mapping to the registry as required.

Example - Setting up Two Single-Partition Instances for Mutual Takeover

The objective for this example is to set up two single-partition database instances with failover support in a mutual takeover configuration. In this example, four servers are configured into two MSCS clusters. By using the mutual takeover configuration, when any of the machine fails, the database server configured for that machine will fail over to the alternative machine, as configured using the MSCS software, and run on the alternative machine.

There are two MSCS clusters in the resulting configuration. Each cluster has:

- Two servers, each with 64 MB of memory and one local SCSI disk of 2 GB
- One SCSI disk tower that has three shared SCSI disks of 2 GB each.

In addition, each machine has one 100X Ethernet Adapter card installed.

Each machine has the following software installed:

- Windows NT Version 4.0 Enterprise Edition with the MSCS feature installed
- DB2 Universal Database Enterprise Edition Version 7.

The resulting network configuration is as follows:

Server 1:	Server 2:
• Machine name:db2test1	• Machine name:db2test2
• TCP/IP hostname:db2test1	• TCP/IP hostname:db2test2
• IP Address: 9.9.9.1 (subnet mask: 255.255.255.0	• IP Address: 9.9.9.2 (subnet mask: 255.255.255.0
• MSCS cluster name: ClusterA	• MSCS cluster name: ClusterA

Both machines in the network are configured with TCP/IP and connected to a private LAN using an Ethernet 100 T-base Hub. In the absence of a Domain Name Server (DNS), all machines have a local TCP/IP `hosts` file, which contains the following entries:

```
9.9.9.1 db2test1 # for Server 1
9.9.9.2 db2test2 # for Server 2
9.9.9.3 ClusterA # for MSCS ClusterA
9.9.9.4 db2tcp1 # for DB2 remote client connection to Server 1
9.9.9.5 db2tcp2 # for DB2 remote client connection to Server 2
```

Preliminary Tasks

Before you perform the following tasks, it is assumed that both machines belong to the same domain, called `DB2NTD`:

1. Create a domain account for DB2 that is a member of the local Administrators group on those machines where DB2 is going to run. Use the account for performing all tasks:
   - Set the user name to db2nt.
   - Set the password to db2nt.
2. Install the MSCS feature on machines db2test1 and db2test2:
   - Name the MSCS cluster ClusterA.
   - The cluster IP Address is 9.9.9.3.
   - Share disk D: will be used by the MSCS software.
   - Share disks E: and F: will be used by DB2.
3. Install DB2 Universal Database Enterprise Edition Version 7 on machine db2test1. Install the software on C:\SQLLIB, which is a local drive.
4. Install DB2 Universal Database Enterprise Edition Version 7 on machine db2test2. Install the software on C:\SQLLIB, which is a local drive.

The next step is to set up the DB2MSCS.CFG file for each instance, and run the DB2MSCS utility for each instance.

Run the DB2MSCS Utility

To set up the db2test1 machine, perform the following tasks:

1. On machine db2test1, log on as user db2nt. The password is db2nt.
2. Create the DB2 instance DB2A, if it does not already exist. The command to create the instance is:

   ```
   db2icrt DB2A
   ```

3. Set up the DB2MSCS.CFG file for the DB2 instance on machine db2test1:

   ```
   #
   #  DB2MSCS.CFG for database system
   #  on machine db2test1
   DB2_INSTANCE=DB2A
   CLUSTER_NAME=ClusterA
   #
   # Group 1
   GROUP_NAME=DB2A Group
   IP_NAME=IP Address for DB2A
   IP_ADDRESS=9.9.9.4
   IP_SUBNET=255.255.255.0
   IP_NETWORK=ClusterA
   NETNAME_NAME=Network name for DB2A
   NETNAME_VALUE=DB2SRV1
   NETNAME_DEPENDENCY=IP Address for DB2A
   DISK_NAME=Disk E:
   INSTPROF_DISK=Disk E:
   ```

4. Run the DB2MSCS utility as follows:

   ```
   db2mscs -f:DB2MSCS.CFG
   ```

5. Log out from the db2nt account.

6. On machine db2test2, log on as user db2nt, which belongs to the local Administrators group. The password is db2nt.
7. Create the DB2 instance DB2B, if it does not already exist. The command to create the instance is:

   ```
   db2icrt DB2B
   ```

8. Set up the DB2MSCS.CFG file for the DB2 instance on machine db2test2:

   ```
   #
   #  DB2MSCS.CFG for database system
   #  on machine db2test2
   DB2_INSTANCE=DB2B
   CLUSTER_NAME=ClusterA
   #
   # Group 1
   GROUP_NAME=DB2B Group
   IP_NAME=IP Address for DB2B
   IP_ADDRESS=9.9.9.5
   IP_SUBNET=255.255.255.0
   IP_NETWORK=ClusterA
   NETNAME_NAME=Network name for DB2B
   NETNAME_VALUE=DB2SRV2
   NETNAME_DEPENDENCY=IP Address for DB2B
   DISK_NAME=Disk F:
   INSTPROF_DISK=Disk F:
   ```

9. Run the DB2MSCS utility as follows:

   ```
   db2mscs -f:DB2MSCS.CFG
   ```

10. Log out from the db2nt account.

Example - Setting up a Four-Node Partitioned Database System for Mutual Takeover

The objective for this example is to set up a four-node partitioned database system with failover support in a mutual takeover configuration. In this example, four servers are configured into two MSCS clusters. By using the mutual takeover configuration, if any machine fails, the database partition servers configured for that machine will fail over to the alternative machine, as configured using the MSCS software, and run as a logical node on the alternative machine.

There are two MSCS clusters in the resulting configuration. Each cluster has:

- Two servers, each with 64 MB of memory and one local SCSI disk of 2 GB
- One SCSI disk tower that has three shared SCSI disks of 2 GB each.

In addition, each machine has one 100X Ethernet Adapter card installed.

Each machine has the following software installed:

- Windows NT Version 4.0 Enterprise Edition with the MSCS feature installed

- DB2 Universal Database Extended Enterprise Edition Version 7.

The resulting network configuration is as follows:

Server 1: • Machine name:db2test1 • TCP/IP hostname:db2test1 • IP Address: 9.9.9.1 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterA	Server 2: • Machine name:db2test2 • TCP/IP hostname:db2test2 • IP Address: 9.9.9.2 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterA
Server 3: • Machine name:db2test3 • TCP/IP hostname:db2test3 • IP Address: 9.9.9.3 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterB	Server 4: • Machine name:db2test4 • TCP/IP hostname:db2test4 • IP Address: 9.9.9.4 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterB

All machines in the network are configured with TCP/IP and connected to a private LAN using an Ethernet 100 T-base Hub. In the absence of a Domain Name Server (DNS), all machines have a local TCP/IP `hosts` file, which contains the following entries:

```
9.9.9.1 db2test1 # for Server 1
9.9.9.2 db2test2 # for Server 2
9.9.9.3 db2test3 # for Server 3
9.9.9.4 db2test4 # for Server 4
9.9.9.5 ClusterA # for MSCS Cluster 1
9.9.9.6 ClusterB # for MSCS Cluster 2
9.9.9.7 db2tcp # for DB2 remote client connection
```

Preliminary Tasks

Before you perform the following tasks, it is assumed that all four machines belong to the same domain, called `DB2NTD`:

1. Create a domain account for DB2 that is a member of the local Administrators group on those machines where DB2 is going to run. Use the account for performing all tasks:
   - Set the user name to `db2nt`.
   - Set the password to `db2nt`.
2. Create a second domain account with the *"password never expires"* characteristic. This account will be associated with DB2 services:
   - Set the user name to `db2mpp`.
   - Set the password to `db2mpp`.
3. Install the MSCS feature on machines `db2test1` and `db2test2`:

- Name the MSCS cluster ClusterA.
- The cluster IP Address is 9.9.9.5.
- Share disk D: will be used by the MSCS software.
- Share disks E: and F: will be used by DB2.

4. Install the MSCS feature on machines db2test3 and db2test4:
   - Name the MSCS cluster ClusterB.
   - The cluster IP Address is 9.9.9.6.
   - Share disk D: will be used by the MSCS software
   - Share disks E: and F: will be used by DB2.
5. Install DB2 Enterprise - Extended Edition on machine db2test1:
   - Select the *"This machine will be the instance-owning database partition server"* option.
   - The account for the DB2 service is db2mpp. The password is db2mpp.
   - Install the software on C:\SQLLIB, which is a local drive.
6. Install DB2 Enterprise - Extended Edition on machines db2test2, db2test3, and db2test4:
   - Select the *"This machine will be a new node on an existing partitioned database system"* option.
   - Select db2test1 as the instance-owning machine.
   - The account for the DB2 service is db2mpp. The password is db2mpp.
   - Install the software on C:\SQLLIB, which is a local drive.

The next step is to set up the DB2MSCS.CFG file and run the DB2MSCS utility.

Run the DB2MSCS Utility

To set up the db2test1 machine, perform the following tasks:

1. Log on as user db2nt, which belongs to the local Administrators group. The password is db2nt.
2. Set up the DB2MSCS.CFG file:

```
#
#  DB2MSCS.CFG for one partitioned database system with
#  multiple MSCS clusters
DB2_INSTANCE=DB2MPP
CLUSTER_NAME=ClusterA
DB2_LOGON_USERNAME=db2mpp
DB2_LOGON_PASSWORD=db2mpp
# Group 1
# for DB2 node 0
GROUP_NAME=DB2NODE0
DB2_NODE=0
IP_NAME=IP Address for DB2
IP_ADDRESS=9.9.9.7
IP_SUBNET=255.255.255.0
IP_NETWORK=Ethernet
```

```
NETNAME_NAME=Network name for DB2
NETNAME_VALUE=DB2WOLF
NETNAME_DEPENDENCY=IP Address for DB2
DISK_NAME=Disk E:
INSTPROF_DISK=Disk E:
#
# Group 2
# for DB2 node 1
GROUP_NAME=DB2NODE1
DB2_NODE=1
DISK_NAME=Disk F:
#
CLUSTER_NAME=ClusterB
# Group 3
# for DB2 node 2
GROUP_NAME=DB2NODE2
DB2_NODE=2
DISK_NAME=Disk E:
#
# Group 4
# for DB2 node 3
GROUP_NAME=DB2NODE3
DB2_NODE=3
DISK_NAME=Disk F:
```

3. Run the DB2MSCS utility as follows:

   ```
   db2mscs -f:DB2MSCS.CFG
   ```

4. Log out from the db2nt account.

The final steps are to register the database drive mapping for the two MSCS clusters.

Register the Database Drive Mapping for ClusterA

To register the database drive mapping for MSCS cluster ClusterA, perform the following tasks:

1. On machine db2test1, log on as user db2mpp, which is the account associated with DB2 services. The password is db2mpp.
2. To register the database drive mapping, enter the following commands:

   ```
   db2drvmp add 0 F E

   db2drvmp add 1 E F
   ```

3. Bring all DB2 resources offline, then bring them online.

Register the Database Drive Mapping for ClusterB

To register the database drive mapping for MSCS cluster ClusterB, perform the following tasks:

1. On machine db2test3, log on as user db2mpp, which is the account associated with DB2 services. The password is db2mpp.
2. To register the database drive mapping, enter the following commands:

```
db2drvmp add 2 F E
db2drvmp add 3 E F
```

3. Bring all DB2 resources offline, then bring them online.

Administering DB2 in an MSCS Environment

If you are using MSCS clusters, your DB2 instance requires additional planning with regards to daily operation, database deployment, and database configuration. For DB2 to execute transparently on any MSCS node, additional administrative tasks must be performed. All DB2 dependent operating system resources must be available on all MSCS nodes. Some of these operating system resources fall outside the scope of MSCS. That is, they cannot be defined as an MSCS resource. You must ensure that each system is configured such that the same operating system resources are available on all MSCS nodes. The sections that follow describe the additional work that must be done.

Starting and Stopping DB2 Resources

You must start and stop DB2 resources from the Cluster Administrator tool. Several mechanisms are available to start a DB2 instance, such as the **db2start** command, and the **Services** option from the Control Panel. However, if DB2 is not started from the Cluster Administrator, the MSCS software will not be aware of the state of the DB2 instance. If a DB2 instance is started using the Cluster Administrator, and stopped using the **db2stop** command, the MSCS software will interpret the **db2stop** command as a software failure, and attempt to restart DB2. (The current MSCS interfaces do not support notification of a *resource state*.)

Similarly, if you use **db2start** to start a DB2 instance, MSCS cannot detect that the resource is online; if a database server fails, MSCS will not bring the DB2 resource online on the failover machine in the cluster.

Three operations can be applied to a DB2 instance:

Online
: This operation is equivalent to using the **db2start** command. If DB2 is already active, this operation can be used simply to notify MSCS that DB2 is active. Any errors during this operation will be written to the Windows NT Event Log.

Offline
: This operation is equivalent to using the **db2stop** command. If there are any active attachments to an instance, this operation will fail. This is consistent with the behavior of **db2stop**.

Fail resource
: This operation is equivalent to using the **db2stop** command with the

force option specified. DB2 will disconnect all applications from the DB2 system, and stop all database servers.

Running Scripts

You can run scripts both before and after a DB2 resource is brought online. These scripts *must* reside in the instance profile directory that is specified for the DB2INSTPROF environment variable. This directory is the directory path that is specified by the "-p" parameter of the **db2icrt** command. You can obtain this value by issuing the following command:

```
db2set -i:instance_name DB2INSTPROF
```

This file path must be on a clustered disk, so that the instance directory is available on all cluster nodes.

These script files are not required, and are only run if they are found in the instance directory. They are launched by the MSCS Cluster Service in the background. The script files must redirect standard output to capture any information returned from commands within the script files. The output is not displayed to the screen.

In a partitioned database environment, by default, the same script will be used by every database partition server in the instance. If you need to distinguish among the different database partition servers in the instance, use different assignments of the DB2NODE environment variable to target specific node numbers (for example, use the IF statement in the `db2cpre.bat` and `db2cpost.bat` files).

Running Scripts Before Bringing DB2 Resources Online

If you want to run a script *before* you bring a DB2 resource online, the script *must* be named `db2cpre.bat`. DB2 calls functions that will launch this batch file from the Windows NT command line processor (CLP) and wait for the CLP to complete execution before the DB2 resource is brought online. You can use this batch file for tasks such as modifying the DB2 database manager configuration. You may want to change some database manager parameter values if the failover system is constrained, and you must reduce the system resources consumed by DB2.

The commands placed in the `db2cpre.bat` script should execute synchronously. Otherwise, the DB2 resource may be brought online before all tasks in the script are completed, which may result in unexpected behavior. Specifically, **db2cmd** should not be invoked in the `db2cpre.bat` script, because it, in turn, launches another command processor, which will run DB2 commands asynchronously to the **db2cmd** program.

If you want to use DB2 CLP commands in the `db2cpre.bat` script, the commands should be placed in a file and run as a CLP batch file from within

a program that initializes the DB2 environment for the DB2 command line processor, and then waits for the completion of the DB2 command line processor. For example:

```
#include <windows.h>

int WINAPI DB2SetCLPEnv_api(DWORD pid);

void main ( int argc, char *argv [ ] )
{
      STARTUPINFO         startInfo   = {0};
      PROCESS_INFORMATION pidInfo     = {0};
      char     title  [32]   = "Run Synchronously";
      char     runCmd [64]  =
                             "DB2 -z c:\\run.out -tvf c:\\run.clp";
/* Invoke API to set up a CLP Environment */
      if ( DB2SetCLPEnv_api (GetCurrentProcessId ()) == 0 ) 1
      {
         startInfo.cb          = sizeof(STARTUPINFO);
         startInfo.lpReserved  = NULL;
         startInfo.lpTitle     = title;
         startInfo.lpDesktop   = NULL;
         startInfo.dwX         = 0;
         startInfo.dwY         = 0;
         startInfo.dwXSize     = 0;
         startInfo.dwYSize     = 0;
         startInfo.dwFlags     = 0L;
         startInfo.wShowWindow = SW_HIDE;
         startInfo.lpReserved2 = NULL;
         startInfo.cbReserved2 = 0;
               if ( CreateProcessA( NULL,
                              runCmd, 2
                              NULL,
                              NULL,
                              FALSE,
                              NORMAL_PRIORITY_CLASS CREATE_NEW_CONSOLE,
                              NULL,
                              NULL,
                              &startInfo,
                              &pidInfo ) )
         {
            WaitForSingleObject (pidInfo.hProcess, INFINITE);
            CloseHandle (pidInfo.hProcess);
            CloseHandle (pidInfo.hThread);
         }
      }
      return;
}
```

1 The API DB2SetCLPEnv_api is resolved by the import library `DB2API.LIB`. This API sets an environment that allows CLP commands to be invoked. If this program is invoked from the `db2cpre.bat` script, the command processor will wait for the CLP commands to complete.

2 runCmd is the name of the script file that contains the DB2 CLP commands.

A sample program called `db2clpex.exe` can be found in the `MISC` subdirectory of the DB2 install path. This executable is similar to the example provided, but accepts the DB2 CLP command as a command line argument. If you want to use this sample program, copy it to the `BIN` subdirectory. You can use this executable in the `db2cpre.bat` script as follows (INSTHOME is the instance directory):

```
db2clpex "DB2 -Z INSTHOME\pre.log -tvf INSTHOME\pre.clp"
```

All DB2 ATTACH commands or CONNECT statements should explicitly specify a user; otherwise, they will be executed under the user account associated with the cluster service. CLP scripts should also complete with the TERMINATE command to end the CLP background process.

Following is an example of a `db2cpre.bat` file:

```
db2cpre.bat : 1
------------------------
db2clpex "db2 -z INSTHOME\pre-%DB2NODE%.log -tvf INSTHOME\pre.clp" 2 - 5
------------------------

PRE.CLP 6
------------------------
update dbm cfg using MAXAGENTS 200;
get dbm cfg;
terminate;
------------------------
```

1 The `db2cpre.bat` script executes under the user account associated with the Cluster Service. If DB2 actions are required, the user account associated with the Cluster Service must be a valid SQL identifier, as defined by DB2.

2 INSTHOME is the instance directory.

3 The name of the log file must be different for each node to avoid file contention when both logical nodes are brought online at the same time.

4 `db2clpex.exe` is a sample program that uses a command line argument to specify the CLP command that is to be invoked.

5 The `db2clpex.exe` sample program must be made available on all MSCS cluster nodes.

6 The CLP commands in this example set a limit on the number of agents.

Running Scripts After Bringing DB2 Resources Online

If you want to run a script *after* you bring a DB2 resource online, it *must* be named `db2cpost.bat`. The script will be run asynchronously from MSCS after the DB2 resource has been successfully brought online. The **db2cmd** command can be used in this script to execute DB2 CLP script files. Use the "-c" parameter of the **db2cmd** command to specify that the utility should close all windows on completion of the task. For example:

```
db2cmd -c db2 -tvf mycmds.clp
```

The "-c" parameter must be the first argument to the **db2cmd** command, because it prevents orphaned command processors in the background.

The `db2cpost.bat` script is useful if you want to perform database activities immediately after the DB2 resource fails over and becomes active. For example, you can restart or activate databases in the instance so that they are primed for user access.

Following is an example of a `db2cpost.bat` script:

```
db2cpost.bat 1
------------------------
db2cmd -c db2 -z INSTHOME\post-%DB2NODE%.log -tvf INSTHOME\post.clp 2 - 4
------------------------

POST.CLP 5
------------------------
restart database SAMPLE;
connect reset;
activate database SAMPLE;
terminate;
------------------------
```

1 The `db2cpost.bat` script runs under the user account associated with the Cluster Service. If DB2 actions are required, the user account associated with the Cluster Service must be a valid SQL identifier, as defined by DB2.

2 INSTHOME is the instance directory.

3 The name of the log file must be different for each node to avoid file contention when both logical nodes are brought online at the same time.

4 The **db2cmd** command can be used, because the `db2cpost.bat` script can run asynchronously. The "-c" parameter must be used to terminate the command processor.

5 The CLP script in this example contains commands to restart and activate the database. This script returns the database to an active state immediately after the database manager is started. In a partitioned database system, you should remove the ACTIVATE

DATABASE command, because multiple DB2 resources are brought online at the same time. The RESTART DATABASE command may fail, because another node is activating the database. If this occurs, rerun the script to ensure that the database is restarted correctly.

Database Considerations

When you create a database, ensure that the database path refers to a share disk. This allows the database to be seen on all MSCS nodes. All logs and other database files must also refer to clustered disks for DB2 to failover successfully. If you do not perform these steps, a DB2 system failure will occur, because it will seem to DB2 that files have been deleted or are unavailable.

Ensure also that the database manager and database configuration parameters are set so that the amount of system resources consumed by DB2 is supported on either MSCS node. The *autorestart* database configuration parameter should be set to `ON`, so that the first database connection on failover will bring the database to a consistent state. (The default setting for *autorestart* is `ON`.) The database can also be brought to a ready state by using the `db2cpost.bat` script to restart and activate the database. This method is preferred, because there will be no dependency on *autorestart*, and the database is brought to a ready state independent of a user connection request.

User and Group Support

DB2 relies on Windows NT for user authentication and group support. For a DB2 instance to fail over from one MSCS node to another in a seamless fashion, each MSCS node must have access to the same Windows NT security databases. You can achieve this by using Windows NT Domain Security.

Define all DB2 users and groups in a Domain Security database. The MSCS nodes must be members of this Domain, or the Domain must be a Trusted Domain. DB2 will then use the Domain Security database for authentication and group support, independent of the MSCS node on which DB2 is running.

If you are using local accounts, the accounts must be replicated on each MSCS node. This approach is not recommended, because it is error prone and requires dual maintenance.

DCE Security is also a supported authentication mode, if all MSCS nodes are clients in the same DCE cell.

You should associate the MSCS service with a user account that follows DB2 naming conventions. This allows the MSCS service to perform actions against DB2 that may be required in the `db2cpre.bat` and `db2cpost.bat` scripts.

For more information about Windows NT user and group support, see ″User Authentication with DB2 for Windows NT″ in the *Administration Guide: Implementation*.

Communications Considerations

DB2 supports two LAN protocols in an MSCS Environment:

- TCP/IP
- NetBIOS

TCP/IP is supported because it is a supported cluster resource type. To enable DB2 to use TCP/IP as a communications protocol for a partitioned database system, create an IP Address resource and place it in the same group as the DB2 resource that represents the database partition server that you want to use as a coordinator node for remote applications. Then create a dependency, using the Cluster Administrator tool, to ensure that the IP resource is online before the DB2 resource is started. DB2 clients can then catalog TCP/IP node directory entries to use this TCP/IP address.

The TCP/IP port associated with the *svcename* database manager configuration parameter must be reserved for use by the DB2 instance on all machines that participate in the instance. The service name associated with the port number must also be the same in the `services` file on all machines.

Although NetBIOS is not a supported cluster resource, you can use NetBIOS as a LAN protocol, because the protocol ensures that NetBIOS names are unique on the LAN. When DB2 registers a NetBIOS name, NetBIOS ensures that the name is not in use on the LAN. In a failover scenario, when DB2 is moved from one system to another, the *nname* used by DB2 will be deregistered from one partner machine in the MSCS cluster and registered on the other machine.

DB2 NetBIOS support uses NetBIOS Frames (NBF). This protocol stack can be associated with different logical adapter numbers (LANA). To ensure consistent NetBIOS access to the server, the LANA associated with the NBF protocol stack should be the same on all clustered nodes. You can configure this by using the **Networks** option from the Control Panel. You should associate NBF with LANA 0, because this is the default setting expected by DB2.

System Time Considerations

DB2 uses the system time to time stamp certain operations. All MSCS nodes that participate in DB2 failover must have the system time zone and system time synchronized to ensure that DB2 behaves consistently on all machines.

Set the system time zone using the **Date/Time** option from the Control Panel dialog. MSCS has a time service that synchronizes the date and time when the

MSCS nodes join to form a cluster. The time service, however, only synchronizes the time every 12 hours, which may result in problems if the time is changed on one system, and DB2 fails over before the time is synchronized.

If the time is changed on one of the MSCS cluster nodes, it should be manually synchronized on the other cluster nodes using the command:

```
net time /set /y \\remote_node
```

Where *remote_node* is the machine name of the cluster node.

Administration Server and Control Center Considerations in a Partitioned Database Environment

The DB2 Administration Server is (optionally) created during the installation of DB2 Universal Database. It is not a partitioned database system. The Control Center uses the services provided by the Administration Server to administer DB2 instances and databases.

In a partitioned database system, a DB2 instance can reside on multiple MSCS nodes. This implies that a DB2 instance must be cataloged on multiple systems under the Control Center so that the instance remains accessible, regardless of the MSCS node on which the DB2 instance is active.

The Administration Server instance directory is not shared. You must mirror all user-defined files in the Administration Server directory to all MSCS nodes to provide the same level of administration to all MSCS nodes. Specifically, you must make user scripts and scheduled executables available on all nodes. You must also ensure that scheduled activities are scheduled on all machines in an MSCS cluster.

Alternatively, instead of duplicating the Administration Server on all machines, you may want to have the Administration Server fail over. For the purposes of the following example, assume that you have two MSCS nodes in the cluster, and that they are called MACH0 and MACH1. MACH0 has access to a cluster disk that will be used by the Administration Server. Assume also that both MACH0 and MACH1 have an Administration Server. You would perform the following steps to make the Administration Server highly available:

1. Stop the Administration Server on both machines by invoking the **db2admin stop** command on each machine.
2. On all administration client machines, uncatalog all references to the Administration Servers on MACH0 and MACH1 using the UNCATALOG NODE command. (You can use the LIST NODE DIRECTORY command on the client machine to determine if any references to the Administration Server exist.)

3. Drop the Administration Server from MACH1 by invoking the **db2admin drop** command from MACH1. (You would only perform this step if you had an Administration Server on both machines.)
4. Determine the name of the Administration Server by issuing the **db2admin** command from MACH0. (The default name is DB2DAS00.)
5. Use the DB2MSCS utility to set up failover support for the Administration Server. This entails creating a DB2 resource on MSCS named DB2DAS00 that has dependencies on the IP and disk resources. (If you have a mutual takeover configuration, you would put the resource in the group that holds the DB2 resource for NODE0.) This resource will be used as the MSCS resource that supports the Administration Server. The DB2MSCS.ADMIN file would be as follows:

```
#
# db2mscs.admin for Administration Server
# run db2mscs -f:db2mscs.admin
#
DB2_INSTANCE=DB2DAS00
CLUSTER_NAME=CLUSTERA
DB2_LOGON_USERNAME=db2admin
DB2_LOGON_PASSWORD=db2admin
# put Administration server in the same group as DB2 Node 0
GROUP_NAME=DB2NODE0 1
DISK_NAME=DISK E:
INSTPROF_DISK=DISK E:
IP_NAME= IP Address for Administration Server
IP_ADDRESS=9.9.9.8
IP_SUBNET=255.255.255.0
IP_NETWORK=Ethernet
```

   1 This group can be the same as the existing group. This way, you do not require an additional disk for the instance profile directory.

6. On MACH1, invoke the following command to set DB2DAS00 as the Administration Server:

```
db2set -g db2adminserver=DB2DAS00
```

7. On MACH0, modify the start-up properties of DB2DAS00 through the Services program so that it is brought up manually and not automatically, because DB2DAS00 is now controlled by MSCS.

When the Administration Server is enabled for failover, all remote access should use an MSCS IP resource for communicating with the Administration Server. The Administration Server will now have the following properties:

- The Administration Server instance directory will fail over with the Administration Server.
- Clients will only catalog a single node to communicate with the Administration Server, regardless of the MSCS node on which it is active.
- Jobs only need to be scheduled once against the Administration Server.

- Local instances can only be controlled by the Administration Server when the Administration Server is active on the same MSCS node as the local instance.
- The Administration Server is not accessible if the Cluster Service is not active.

Limitations and Restrictions

When you run DB2 in an MSCS environment:

- You cannot use physical I/O on shared disks, unless the shared disks have the same physical disk number across both MSCS nodes. You can use logical I/O, because the disk is accessed using a partition identifier.
- You must configure all DB2 resource for MSCS support. If you do not, system errors will occur during DB2 run time (DB2 cannot properly operate in the absence of system resources). For example, if the database logs are not on an MSCS shared disk, DB2 cannot restart the database.
- You must manage a DB2 instance from the Cluster Administrator tool. MSCS will view other mechanisms that are used to start and stop the database manager as software inconsistencies. For example, if you use MSCS to start DB2, and the **db2stop** command to stop DB2, MSCS will detect this as a software failure, and will restart the instance. This also means that you should not use the Control Center to start and stop DB2.
- To uninstall DB2, you must first stop MSCS.

Chapter 8. High Availability in the Solaris Operating Environment

High availability in the Solaris Operating Environment can be achieved through DB2 working with Sun Cluster 2.x (SC2.x), Sun Cluster 3.0 (SC3.0), or Veritas Cluster Server (VCS). For information about Sun Cluster 3.0, see the white paper entitled "DB2 and High Availability on Sun Cluster 3.0", which is available from the "DB2 UDB and DB2 Connect Online Support" Web site (http://www.ibm.com/software/data/pubs/papers/). For information about VERITAS Cluster Server, see the white paper entitled "DB2 and High Availability on VERITAS Cluster Server", which is also available from the "DB2 UDB and DB2 Connect Online Support" Web site.

This chapter describes in detail how DB2 works with Sun Cluster 2.x (SC2.x) 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 31).

Figure 31. DB2, Sun Cluster 2.x, and High Availability

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 32. The private network interfaces are used to send *heartbeat* messages, as well as

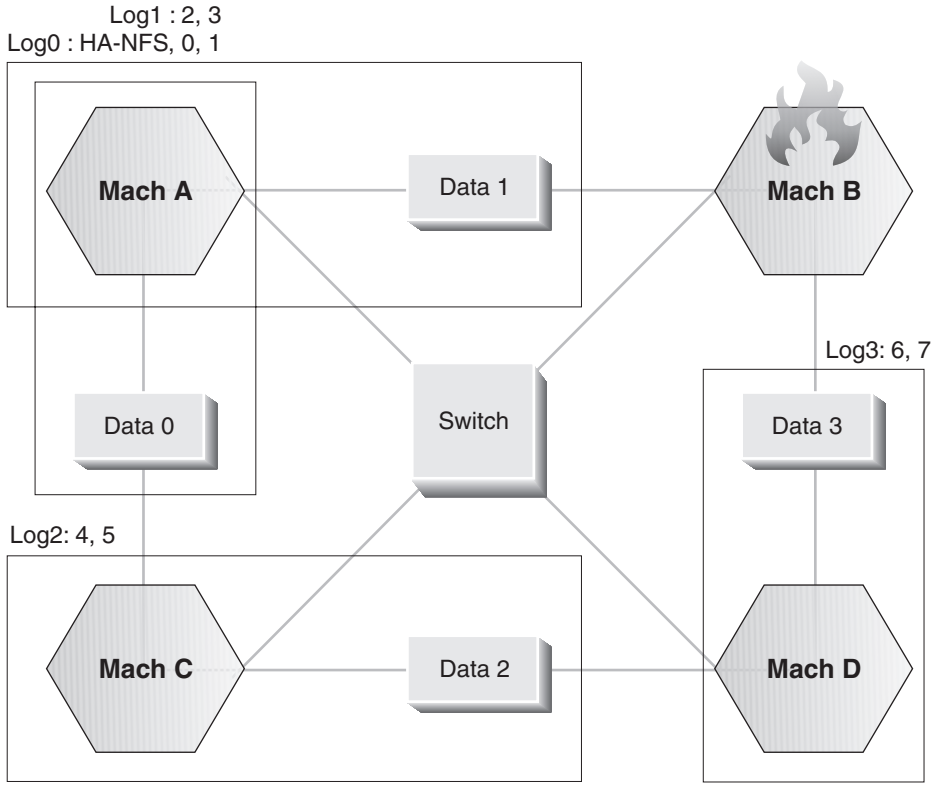


Figure 32. Failover

control messages, among the machines in the cluster. The public network interfaces are used to communicate directly with clients of the HA cluster. The disks in an HA cluster are connected to two or more machines in the cluster, so that if one machine fails, another machine has access to them.

A data service running on an HA cluster has one or more logical public network interfaces and a set of disks associated with it. The clients of an HA data service connect via TCP/IP to the logical network interfaces of the data service only. If a failover occurs, the data service, along with its logical network interfaces and set of disks, are moved to another machine.

One of the benefits of an HA cluster is that a data service can recover without the aid of support staff, and it can do so at any time. Another benefit is redundancy. All of the parts in the cluster should be redundant, including the machines themselves. The cluster should be able to survive any single point of failure.

Even though highly available data services can be very different in nature, they have some common requirements. Clients of a highly available data service expect the network address and host name of the data service to remain the same, and expect to be able to make requests in the same way, regardless of which machine the data service is on.

Consider a Web browser that is accessing a highly available Web server. The request is issued with a URL (Uniform Resource Locator), which contains both a host name, and the path to a file on the Web server. The browser expects both the host name and the path to remain the same after a failover ofáthe Web server. If the browser is downloading a file from the Web server, and the server is failed over, the browser will need to reissue the request.

Availability of a data service is measured by the amount of time the data service is available to its users. The most common unit of measurement for availability is the percentage of ″up time″; this is often referred to as the number of ″nines″:

```
99.99% => service is down for (at most) 52.6 minutes / yr
99.999% => service is down for (at most) 5.26 minutes / yr
99.9999% => service is down for (at most) 31.5 seconds / yr
```

When designing and testing an HA cluster:

1. Ensure that the administrator of the cluster is familiar with the system and what should happen when a failover occurs.
2. Ensure that each part of the cluster is truly redundant and can be replaced quickly if it fails.
3. Force a test system to fail in a controlled environment, and make sure that it fails over correctly each time.
4. Keep track of the reasons for each failover. Although this should not happen often, it is important to address any issues that make the cluster unstable. For example, if one piece of the cluster caused a failover five times in one month, find out why and fix it.
5. Ensure that the support staff for the cluster is notified when a failover occurs.
6. Do not overload the cluster. Ensure that the remaining systems can still handle the workload at an acceptable level after a failover.
7. Check failure-prone components (such as disks) often, so that they can be replaced before problems occur.

Fault Tolerance and Continuous Availability

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.

Continuous availability is a step above high availability. It shelters its clients from both planned and unplanned down time. With a continuous availability configuration, the client is completely unaffected if one of the machines hosting the data service fails or is brought down for maintenance. Continuous availability configurations are complex and more expensive to implement.

An HA cluster is the most common solution to increase availability because it is scalable, easy to use, and relatively inexpensive to implement.

Sun Cluster 2.2

Sun Cluster 2.2 (SC2.2) is Sun Microsystems' clustering and high availability product. SC2.2 supports up to four machines in a single cluster. Using four Ultra Enterprise 10000s, a cluster can have up to 256 CPUs and 256 GB of RAM.

Supported Systems

System	UltraSPARC	Memory Capacity	I/O
Ultra Enterprise 1	1	64MB-1GB	3 SBus
Ultra Enterprise 2	1-2	64MB-2GB	4 SBus
Ultra Enterprise 450	1-4	32MB-4GB	10 PCI
Ultra Enterprise 3000	1-6	64MB-6GB	9 SBus
Ultra Enterprise 4000	1-14	64MB-14GB	21 SBus
Ultra Enterprise 5000	1-14	64MB-14GB	21 SBus
Ultra Enterprise 6000	1-30	64MB-30GB	45 SBus
Ultra Enterprise 10000	1-64	512MB-64GB	64 SBus

Agents

The Sun Cluster software includes a number of high availability agents that are supported and shipped with the SC2.2 product. Other HA agents, such as the one for DB2, are developed outside of Sun, and are not shipped with the Sun Cluster software. The HA agent for DB2 is shipped with DB2, is supported by IBM, and is supplied free of charge with DB2.

The Sun Cluster software works with highly available data services by providing an opportunity to register methods (scripts or programs) that correspond to various components of the Sun Cluster software. Utilizing these methods, the SC2.2 software can control a data service without having intimate knowledge of it. These methods include:

START
Used to start portions of the data service before the logical network interfaces are online.

START_NET
Used to start portions of the data service after the logical network interfaces are online.

STOP Used to stop portions of the data service after the logical network interfaces are offline.

STOP_NET
Used to stop portions of the data service before the logical network interfaces are offline.

ABORT
Like the STOP method, except it is run just before a machine is brought down by the cluster software. In this case, the machine's ″health″ is in question, and a data service may want to execute ″last wish″ requests before the machine is brought down. Run after the logical network interfaces are offline.

ABORT_NET
Like the ABORT method, except it is run before the logical network interfaces are offline.

FM_INIT
Used to initialize fault monitors.

FM_START
Used to start the fault monitors.

FM_STOP
Used to stop the fault monitors.

FM_CHECK
Called by the **hactl** command. Returns the current status of the corresponding data service.

The DB2 agent consists of the following scripts: START_NET, STOP_NET, FM_START, and FM_STOP. The following scripts are not run during cluster reconfiguration: ABORT, ABORT_NET, and FM_CHECK.

A high availability agent consists of one or more of these methods. The methods are registered with SC2.2 through the **hareg** command. Once registered, the Sun Cluster software will call the corresponding method to control the data service.

It is important to remember that the ABORT and STOP methods of a service may not be called. These methods are intended for the controlled shutdown of a data service, and the data service must be able to recover if a machine fails without calling them.

For more information, refer to the Sun Cluster documentation.

Logical Hosts

The SC2.2 software uses the concept of a logical host. A *logical host* consists of a set of disks and one or more logical public network interfaces. A highly available data service is associated with a logical host, and requires the disks that are in the disk groups of the logical host. Logical hosts can be hosted by different machines in the cluster, and ″borrow″ the CPUs and memory of the machine on which they are running.

Logical Network Interfaces

As with other UNIX based operating systems, Solaris has the ability to have extra IP addresses, in addition to the primary one for a network interface. The extra IP addresses reside on a logical interface in the same way that the primary IP address resides on the physical network interface. Following is an example of the logical interfaces on two machines in a cluster. There are two logical hosts, and both are currently on the machine ″thrash″.

```
scadmin@crackle(202)# netstat -in
Name Mtu Net/Dest Address Ipkts Ierrs Opkts Oerrs Collis Queue
lo0 8232 127.0.0.0 127.0.0.1 289966 0 289966 0 0 0
hme0 1500 9.21.55.0 9.21.55.98 121657 6098 764122 0 0 0
scid0 16321 204.152.65.0 204.152.65.1 489307 0 476479 0 0 0
scid0:1 16321 204.152.65.32 204.152.65.33 0 0 0 0 0 0
scid1 16321 204.152.65.16 204.152.65.17 347317 0 348073 0 0 0
```

1. lo0 is the loopback interface
2. hme0 is the public network interface (ethernet)
3. scid0 is the first private network interface (SCI or Scalable Coherent Interface)
4. scid0:1 is a logical network interface that the Sun Cluster software uses internally
5. scid1 is the second private network interface

```
scadmin@thrash(203)# netstat -in
Name Mtu Net/Dest Address Ipkts Ierrs Opkts Oerrs Collis Queue
```

```
lo0 8232 127.0.0.0 127.0.0.1 1128780 0 118780 0 0 0
hme0 1500 9.21.55.0 9.21.55.92 1741422 5692 757127 0 0 0
hme0:1 1500 9.21.55.0 9.21.55.109 0 0 0 0 0 0
hme0:2 1500 9.21.55.0 9.21.55.110 0 0 0 0 0 0
scid0 16321 204.152.65.0 204.152.65.2 476641 0 489476 0 0 0
scid0:1 16321 204.152.65.32 204.152.65.34 0 0 0 0 0 0
scid1 16321 204.152.65.16 204.152.65.18 348199 0 347444 0 0 0
```

1. hme0:1 is a logical network interface for a logical host
2. hme0:2 is a logical network interface for another logical host

A logical host can have one or more logical interfaces associated with it. These logical interfaces move with the logical host from machine to machine, and are used to access the data service that is associated with the logical host. Because these logical interfaces move with the logical hosts, clients can access the data service independently of the machine on which it resides.

A highly available data service should bind to the TCP/IP address INADDR_ANY. This ensures that each IP address on the system can accept connections for the data service. If a data service binds to a specific IP address instead, it must bind the logical interface associated with the logical host that is hosting the data service. Binding to INADDR_ANY also removes the need to rebind to a new IP address if one arrives on the system that is needed by the data service.

Note: Clients of an HA instance should catalog the database using the host name for the logical IP address of a logical host. They should never use the primary host name for a machine, because there is no guarantee that DB2 will be running on that machine.

Disk Groups and File Systems

Disks for a data service are associated with a logical host in groups (or sets). If the cluster is running Sun StorEdge Volume Manager (Veritas), the Sun Cluster software uses the Veritas ″vxdg″ utility to import and deport the disk groups for each logical host. Following is an example of the disk groups for two logical hosts, ″log0″ and ″log1″, which are being hosted by a machine called ″thrash″. The machine called ″crackle″ is not currently hosting any logical hosts.

```
scadmin@crackle(206)# vxdg list
NAME STATE ID
rootdg enabled 899825206.1025.crackle

scadmin@thrash(205)# vxdg list
NAME STATE ID
rootdg enabled 924176206.1025.thrash
data0 enabled 925142028.1157.crackle=
data1 enabled 899826248.1108.crackle
```

The disk groups ″data0″ and ″data1″ correspond to the logical hosts ″log0″ and ″log1″, respectively. The disk group ″data0″ can be deported from ″thrash″ by running

```
vxdg deport data0
```

and imported to ″crackle″ by running

```
vxdg import data1
```

This is done automatically by the Sun Cluster software, and should not be done manually on a live cluster.

Each disk group contains a number of disks that can be shared between two or more machines in the cluster. A logical host can only be moved to another machine that has physical access to the disks in the disk groups that belong to it.

There are two files that control the file systems for each logical host:

```
/etc/opt/SUNWcluster/conf/hanfs/vfstab.<logical_host>
/etc/opt/SUNWcluster/conf/hanfs/dfstab.<logical_host>
```

where *logical_host* is the name of the associated logical host name.

The `vfstab` file is similar to the `/etc/vfstab` file, except that it contains entries for the file systems to be mounted after the disk groups have been imported for a logical host. The `dfstab` file is similar to the `/etc/dfs/dfstab` file, except that is contains entries for file systems to export through HA-NFS for a logical host. Each machine has its own copy of these files, and care should be taken to ensure that they have the same content on each machine in the cluster.

Note: The paths for the `vfstab` and `dfstab` files of a logical host are misleading, because they contain the directory `hanfs`. Only the `dfstab` file for a logical host is used for HA-NFS. The `vfstab` file is used, even if HA-NFS is not configured.

Following are examples from a cluster running DB2 Universal Database Enterprise - Extended Edition (EEE) in a mutual takeover configuration:

```
scadmin@thrash(217)# ls -l /etc/opt/SUNWcluster/conf/hanfs
total 8
-rw-r--r-- 1 root build 173 Apr 14 15:01 dfstab.log0
-rw-r--r-- 1 root build 316 Apr 26 12:07 vfstab.log0
-rw-r--r-- 1 root build 389 Apr 13 21:04 vfstab.log1

scadmin@thrash(218)# cat dfstab.log0
share -F nfs -o root=crackle:thrash:\
jolt:bump:crackle.torolab.ibm.com:thrash.torolab.ibm.com:\
jolt.torolab.ibm.com:bump.torolab.ibm.com /log0/home
```

The hosts, which are given permission to mount the file system, /log0/home, are from all of the network interfaces (logical and physical) on each machine in the cluster. The file systems are exported with root permissions.

```
scadmin@thrash(220)# cat vfstab.log0
#device to mount             device  to fsck               mount
#                                                          point

/dev/vx/dsk/data0/data1-stat /dev/vx/rdsk/data0/data1-stat /log0
/dev/vx/dsk/data0/vol01      /dev/vx/rdsk/data0/vol01      /log0/home
/dev/vx/dsk/data0/vol02      /dev/vx/rdsk/data0/vol02      /log0/data

scadmin@thrash(221)# cat vfstab.log1
#device to mount             device  to fsck               mount
#                                                          point
/dev/vx/dsk/data1/data1-stat /dev/vx/rdsk/data1/data1-stat /log1
/dev/vx/dsk/data1/vol01      /dev/vx/rdsk/data1/vol01      /log1/home
/dev/vx/dsk/data1/vol02      /dev/vx/rdsk/data1/vol02      /log1/data
/dev/vx/dsk/data1/vol03      /dev/vx/rdsk/data1/vol03      /log1/data1



FS   fsck mount   options
type pass at boot

ufs  2    no      -
ufs  2    no      -
ufs  2    no      -


FS   fsck mount   options
type pass at boot

ufs  2    no      -
ufs  2    no      -
ufs  2    no      -
ufs  2    no      -
```

The vfstab.log0 file contains three valid entries for file systems under the /log0 directory. Notice that the file systems for the logical host log0 use logical volume devices, which are part of the disk group data0 that is associated with the logical host.

The file systems in the vfstab files are mounted in order from top to bottom, so it is important to ensure that the file systems are listed in the correct order. File systems that are mounted underneath a particular file system should be listed below it. The actual file systems that are needed for a logical host depend on the needs of the data service, and will vary considerably from these examples.

During a failover, the SC2.2 software is responsible for ensuring that the disk groups and logical interfaces associated with a logical host follow it around

the cluster from machine to machine. The highly available data service expects to have at least these resources available on a new system after a failover. In fact, many data services are not even aware that they are highly available, and must have these resources ″appear″ to be exactly the same after a failover.

Control Methods

The control methods are registered using

```
hareg(1m)
```

Once an HA service is registered, SC2.2 is responsible for calling the methods that were registered for the HA service at appropriate times during a cluster reconfiguration or failover.

The following actions take place (in the given order) during a cluster reconfiguration (controlled failover). Actions preceding step 5c will not be taken if a machine crashes. (For more information about cluster reconfiguration, refer to the SC2.2 documentation.)

```
 1. FM_STOP method is run.
 2. STOP_NET method is run.
 3. Logical interfaces for the logical host are brought offline.
      - ifconfig hme0:1 0.0.0.0 down
 4. STOP method is run.
 5. Disk groups and file systems are moved.
      a. Unmount logical host file systems.
      b. vxdg deport disk groups on one machine.

    - - Only the steps below are run if a machine crashes - -

      c. vxdg import disk groups on the other machine.
      d. fsck logical host file systems.
      e. Mount logical host file systems.
 6. START method is run.
 7. Logical interfaces for the logical host are brought online.
      - ifconfig hme0:1 <ip address> up
 8. START_NET method is run.
 9. FM_INIT method is run.
10. FM_START method is run.
```

The control methods are run with the following command line arguments:

```
METHOD <logical hosts being hosted> <logical hosts not being hosted> <time-out>
```

The first argument is a comma delimited list of logical hosts that are currently being hosted, and the second is a comma delimited list of logical hosts that are not being hosted. The last argument is the time-out for the method, the amount of time that the method is allowed to run before the SC2.2 software aborts it.

Disk and File System Configuration

SC2.2 supports two volume managers: Sun StorEdge Volume Manager (Veritas) and Solstice Disk Suite. Although both work well, the StorEdge Volume Manager has some advantages in a clustered environment. In some cluster configurations, the controller number for a disk enclosure can be different for each machine in the cluster. If the controller number is different, the paths for the disk devices for the controller will also be different. Because Disk Suite works directly with the disk device paths, it will not work well in this situation. The StorEdge Volume Manager works with the disks themselves, regardless of the controller number, and is not affected if the controller numbers are different.

Since the goal of HA is to increase availability for a data service, it is important to ensure that all file systems and disk devices are mirrored, or in a RAID configuration. This will prevent failovers due to a failed disk, and increase the stability of the cluster.

HA-NFS

DB2 UDB EEE requires a shared file system when an instance is configured across multiple machines. A typical DB2 UDB EEE configuration has the home directory exported from one machine through NFS, and mounted on all of the machines participating in the EEE instance. For a mutual takeover configuration, DB2 UDB EEE depends on HA-NFS to provide a shared, highly available file system. One of the logical hosts exports a file system through HA-NFS, and each machine in the cluster then mounts the file system as the home directory of the EEE instance. For more information about HA-NFS, refer to the Sun Cluster documentation.

The cconsole and ctelnet Utilities

Two useful utilities that come with SC2.2 are `cconsole` and `ctelnet`. These utilities can be used to issue a single command to several machines in a cluster simultaneously. Editing a configuration file with these utilities ensures that it will remain identical on all of the machines in the cluster. These utilities can also be used to install software in exactly the same way on each machine. For more information about these utilities, refer to the Sun Cluster documentation.

Campus Clustering and Continental Clustering

A cluster is called a *campus cluster* when its machines are not in the same building. A campus cluster is useful for removing the building itself as the single point of failure. For example, if the machines in the cluster are all in the same building, and it burns down, the entire cluster is affected. However, if the machines are in different buildings, and one of the buildings burns down, the cluster survives.

A *continental cluster* is a cluster whose machines are distributed among different cities. In this case, the goal is to remove the geographic region as the single point of failure. This type of cluster provides protection against catastrophic events, such as earthquakes and tidal waves.

Currently, a Sun Cluster can support machines as far apart as 10 km, or about 6 miles. This makes campus clustering a viable option for those who need high speed connections between two different sites. A cluster requires two private interconnects, and a number of fiber optic cables for the shared disks. The cost of high speed connections between two sites may offset the benefits.

Common Problems

The SC2.2 software uses the Cluster Configuration Database, or CCD(4), to provide a single cluster-wide repository for the cluster configuration. The CCD has a private API and is stored under the `/etc/opt/SUNWcluster/conf` directory. In rare cases, the CCD can go out of synchrony, and may need to be repaired. The best way to repair the CCD in this situation is to restore it from a backup copy.

To back up the CCD, shut down the cluster software on all machines in the cluster, "tar" up the `/etc/opt/SUNWcluster/conf` directory, and store the tar file in a safe place. If the cluster software is not shut down when the backup is made, you may have trouble restoring the CCD. Ensure that the backup copy is kept up-to-date by taking a fresh backup any time that the cluster configuration is changed. To restore the CCD, shut down the cluster software on all machines in the cluster, move the `conf` directory to `conf.old`, and "untar" the backup copy. The cluster can then be started with the new CCD.

DB2 Considerations

The following topics are covered in this section:

Applications Connecting to an HA Instance

Applications that rely on a highly available DB2 instance must be able to reconnect in the event of a failover. Since the host name and IP address of a logical host remain the same, there is no need to connect to a different host name or to recatalog the database.

Consider a cluster with two machines and one DB2 Universal Database Enterprise Edition (EE) instance. The EE instance will normally reside on one of the machines in the cluster. Clients of the HA instance will connect to the logical IP address (or host name) of the logical host associated with the HA instance.

According to an HA client, there are two types of failover. One type occurs if the machine that is hosting the HA instance crashes. The other type occurs when the HA instance is given an opportunity to shut down gracefully.

If a machine crashes and takes down the HA instance, both existing connections and new connections to the database will hang. The connections hang because there are no machines on the network with the IP address that the clients were using for the database. If the database is shut down gracefully, a `db2stop force` breaks existing connections to the database, and an error message is returned.

During the failover, the logical IP address associated with the database is offline, either because the SC2.2 software took it offline, or because the machine that was hosting the logical host crashed. At this point, any new connections to the database will hang for a short period of time.

The logical IP address associated with the database is eventually brought up on another machine before DB2 is started. At this stage, a connection to the database will not hang, but will receive a communication error, because DB2 has not yet been started on the system. DB2 clients that were still connected to the database will also begin receiving communication errors. Although the clients still believe they are connected, the machine that has started hosting the logical IP address has no knowledge of any existing connections. The connections are simply reset, and the DB2 client receives a communication error. After a short time, DB2 will be started on the machine, and a successful connection to the database can be made. At this point, the database may be inconsistent, and clients may have to wait for it to recover.

When designing an application for an HA environment, it is not necessary to write special code for the stages where the database connections hang. The connections only hang for a short period of time while the Sun Cluster software moves the logical IP address. Any data service running on Sun Cluster will experience the same hanging connections during this stage. No matter how the database comes down, the clients will receive an error

message, and must try to reconnect until successful. From the client's perspective, it is as if the HA instance went down, and was brought back up on the same machine. In a controlled failover, it appears to the client that it was forced off, and that it can later reconnect to the database on the same machine. In an uncontrolled failover, it appears to the client that the database server crashed, and was soon brought back up on the same machine.

Disk Layout for EE and EEE Instances

DB2 expects the disk devices or file systems it requires to appear the same on each machine in the cluster. To ensure that this happens, the required disks or file systems should be configured in such a way that they follow the logical host associated with the HA instance, and will have the same path names on each machine in the cluster.

Both DMS and SMS table spaces are supported in an HA environment. Device containers for DMS table spaces must use raw devices created by the volume manager, which are either mirrored, or in a RAID configuration. Regular disk devices, such as `/dev/rdsk/c20t0d0s0` should not be used because:

- It increases the possibility that the device could be written to from more than one machine at the same time.
- The controller number may be different on another machine.

If DB2 is failed over in this situation, the disk devices it requires will not look the same as they did on the other machine, and it will not start. File containers for DMS table spaces, and containers for SMS table spaces, must reside on mounted file systems. The file systems for a logical host are mounted automatically when they are included in the `vfstab` file for the logical host.

The `vfstab` file for a logical host is in the path:

```
/etc/opt/SUNWcluster/conf/hanfs/vfstab.<logical_host>
```

where *logical_host* is the name of the logical host that is associated with the `vfstab` file.

Each logical host has its own `vfstab` file, which contains file systems that are to be mounted after the disk groups for the logical host have been transferred to the current machine, but before the HA services are started. The Sun Cluster software will try to mount any file system that is properly defined after running **fsck** (file system check), to ensure the health of the file system. If **fsck** fails, the file system will not be mounted, and an error message is logged.

Note: If a process has an open file, or its current working directory is under a mount point, the mount will fail. To prevent this, ensure that no processes are left under the mount points contained in the logical host's `vfstab` file.

Any convention can be used for the file system layout of an EEE instance when using SMS table spaces. Following is the convention used by the hadb2_setup utility:

```
scadmin@crackle(190)# pwd
/export/ha_home/db2eee/db2eee
scadmin@crackle(191)# ls -l
total 18
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0000 -> /log0/disks/db2eee/NODE0000
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0001 -> /log0/disks/db2eee/NODE0001
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0002 -> /log0/disks/db2eee/NODE0002
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0003 -> /log0/disks/db2eee/NODE0003
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0004 -> /log0/disks/db2eee/NODE0004
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0005 -> /log1/disks/db2eee/NODE0005
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0006 -> /log1/disks/db2eee/NODE0006
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0007 -> /log1/disks/db2eee/NODE0007
lrwxrwxrwx 1 root build 28 Aug 12 19:08 NODE0008 -> /log1/disks/db2eee/NODE0008
scadmin@crackle(192)#
```

The instance owner is `db2eee`, and the default database directory for the `db2eee` instance is `/export/ha_home/db2eee`. Logical host `log0` is hosting database partitions 0, 1, 2, and 3, while logical host `log1` is hosting database partitions 4, 5, 6, 7, and 8.

For each database partition, there is a corresponding `NODE`*`xxxx`* directory. The node directories for the database partitions point to a directory under the associated logical host file system.

When choosing a path convention, ensure that:

1. The disks for the file system are in a disk group of the logical host responsible for the database partitions that need them.
2. The file systems that hold containers are mounted through the logical host's `vfstab` file.

Home Directory Layout for EE and EEE Instances

For an EE instance, the home directory should be a file system that is defined in the `vfstab` file for a logical host. This directory will be available before DB2 is started, and is transferred with DB2 to wherever the logical host is moved in the cluster. Each machine has its own copy of the `vfstab` file, and care should be taken to ensure that it has the same contents on each machine. Following is an example of the home directory for an EE instance:

```
/log0/home/db2ee
```

where /log0 is the logical host file system for the logical host log0, and db2ee is the name of the DB2 instance. This home directory path should be placed in the /etc/passwd file on each machine in the cluster that could host the "db2ee" instance.

For an EEE instance, there are two ways to set up the home directory. For a hot standby configuration, the home directory can be set up in the same way as for an EE instance. For a mutual takeover configuration, HA-NFS must be used for the home directory, and must be configured properly *before* setting up the EEE instance.

One of the machines in the cluster must export the file system for the EEE instance, using the dfstab file for a chosen logical host. The dfstab file contains file systems that should be exported through NFS when a machine is hosting a logical host. Each machine has its own copy of the dfstab file, and care should be taken to ensure that it has the same contents on each machine.

Information for the HA-NFS file system is placed in the hadb2tab file (through the hadb2_setup program). When an HA agent reads the information for the instance, it automatically mounts the HA-NFS file system for the instance (see "The hadb2tab File" on page 271).

The mount point for the HA-NFS file system is typically /export/ha_home. On each machine in the cluster, this would be NFS mounted from the logical host that is exporting the HA-NFS directory. The EEE instance owner's home directory is placed under this directory and is called:

```
/export/ha_home/<instance>
```

where *instance* is the name of the instance owner.

One could have a home directory for an instance on each machine, to avoid having to mount or unmount it. Doing this requires extra administrative overhead to ensure that the home directories remain identical on each machine. Failure to do so can prevent DB2 from starting properly, or cause it to start with a different configuration. This is *not* a supported configuration.

Logical Hosts and DB2 UDB EEE

A logical host is usually chosen to host one or more database partitions, as well as export the HA-NFS file system. For example, if there are four database partitions and two machines in the cluster, there should be one logical host for each machine (Figure 33 on page 269). One logical host could host two database partitions, and export the HA-NFS file system, while the other logical host could host the remaining two database partitions.

By default, a DB2 UDB EEE instance allocates enough resources to successfully add up to two database partitions to a machine that already has

one or more live database partitions for that instance. For example, if there are four database partitions for a single instance on a cluster, this will only be a concern if there is one database partition per logical host, or one logical host is hosting three database partitions. In either case, it is possible to have three database partitions fail over to a machine that is already hosting a database partition for the same instance.

The DB2_NUM_FAILOVER_NODES registry variable can be used to increase the amount of resource reserved for database partitions that are failed over.

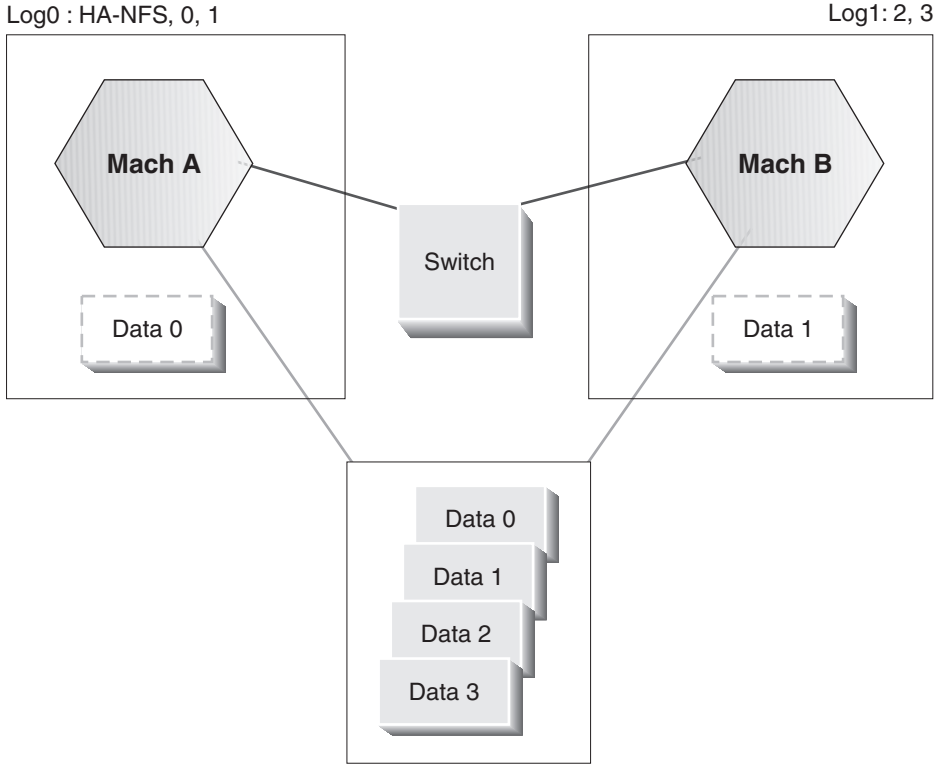


Figure 33. One Logical Host For Each Machine

DB2 Installation Location and Options

The file system on which DB2 is installed should be mirrored, or at least be in a RAID configuration. If DB2 is installed on regular disks, disk failure is more likely; the resulting failover is considered preventable, and decreases the stability of the cluster.

DB2 cannot be installed on disks in a disk group for a logical host, because the HA agent always needs to have access to the DB2 libraries. If the HA

agents do not have access to the DB2 libraries, they will fail. DB2 must be installed normally on each machine in the cluster.

Database and Database Manager Configuration Parameters

The database manager configuration parameters can be changed after a failover, and before DB2 is started, by using the `pre_db2start` script (see "User Scripts" on page 274). This executable script is run (if it exists) under the `sqllib/ha` directory of the instance owner's home directory. As the name suggests, it is run just before **db2start**. The same arguments that are passed to the control methods are passed to the `pre_db2start` script, unless the instance is an EEE instance. For an EEE instance, the `pre_db2start` script is also passed the node number for the **db2start** command.

Crash Recovery

Crash recovery in an HA environment is the same as it would in a regular environment. Even if the HA instance is brought up on a different machine from the one on which it crashed, the files and disk devices for the instance will look the same, and the actions needed to recover the database will not be different. For more information about crash recovery and other forms of database recovery, see "Chapter 1. Developing a Good Backup and Recovery Strategy" on page 3.

Although a database can be restarted manually (or through one of the user scripts), it is recommended that the *autorestart* database configuration parameter be set to `ON`, especially for an EEE instance. This will minimize the amount of time that the database is in an inconsistent state.

High Availability through Data Replication

Data availability can also be enhanced through replication. By replicating data between two servers, a form of high availability is achieved. If one of the servers goes down, the other server should be able to take over and continue to provide the data service.

However, because the replication is done asynchronously, some changes may not have been propagated to the other server when that server goes down.

DB2 Connect Prerequisites on Sun Cluster 2.2

DB2 Connect is supported on Sun Cluster 2.2 if:

- The protocol to the host is TCP/IP (not SNA)
- Two-phase commit is not used. This restriction is relaxed if the user configures the SPM log to be on a shared disk (this can be done through the *spm_log_path* database manager configuration parameter), and the failover machine has an identical TCP/IP configuration (the same host name, IP address, and so on).

The DB2 High Availability Agent

The DB2 high availability agent acts like a mediator between DB2 and SC2.x. It provides a way for the Sun Cluster 2.2 software to control DB2 in a clustered environment, without having intimate knowledge of DB2. There is one agent for both EE and EEE instances. The agent supports both administrative instances and database instances.

Registering the hadb2 Service

To work with SC2.2, the DB2 HA agent must be registered. Registering a data service tells SC2.2 which control methods are available, and in which directory they reside. A special script called `hadb2_reg`, which is shipped with the HA agent, can register the `hadb2` service for both EE and EEE instances. The `hadb2_reg` script needs to be run only once for the entire cluster.

Although there is only one set of control methods for the DB2 HA agent, the way they are registered depends on whether or not an EEE instance will be used in a mutual takeover configuration. For an EE instance or EEE instance in a hot standby configuration, HA-NFS is not used; therefore, the `"-d nfs"` switch, which tells the SC2.2 software that the `hadb2` service is dependent on HA-NFS, is not needed.

The actual command that `hadb2_reg` uses to register the DB2 V7.1 control methods for an EEE instance is:

```
hareg -r hadb2 -b /opt/IBMdb2/V7.1/ha -m
START=hadb2_start,START_NET=hadb2_startnet,STOP_NET=hadb2_stopnet,
FM_START=hadb2_fmstart,FM_STOP=hadb2_fmstop
-t START_NET=$TIMEOUT,STOP_NET=$TIMEOUT -d nfs
```

The `-b` switch tells SC2.x to look in the `opt/IBMdb2/V7.1/ha` directory for all of the control methods. The `-m` switch defines the actual control methods for the `hadb2` service. The `-t` switch defines the timeout for the START_NET and STOP_NET control methods. For a detailed description of each control method, refer to the Sun Cluster documentation.

The `hadb2_unreg` script can be used to unregister the `hadb2` service and, like `hadb2_reg`, needs to be run only once for the cluster.

The hadb2tab File

The `hadb2tab` file is the main configuration file for the DB2 HA agent. Each control method consults this file to find out which instances are highly available. The `hadb2tab` file is located under the `/var/db2/v71/` directory for DB2 UDB Version 7.1. The file supports multiple instances, and each non-commented line represents a different HA instance. Following is an example of an `hadb2tab` file:

```
<scadmin@thrash(203)# cat hadb2tab
EEE DATA db2eee jolt ON /export/ha_home /log0/home #Added by DB2 HA software
EE ADMIN db2ee log1 ON  -               -          #Added by DB2 HA software
```

The first field indicates to the DB2 HA agent whether the instance is an EE instance, or an EEE instance. The second field indicates whether the instance is a data instance, or an administrative instance. The third field contains the user name of the HA instance. The fourth field is the logical host or the HA-NFS host for the instance, depending on whether it is an EE or an EEE instance. The fifth field indicates whether fault monitoring for the instance is turned on or off. The last two fields are the local mount point, and the remote HA-NFS directory, respectively. These fields should be set to - (hyphen) if they are not used, and should only be used with an EEE mutual takeover configuration. Comments are allowed in the `hadb2tab` file if the information on the line before a "#" marker is either of zero length, or a valid definition of an instance.

Control Methods

Control methods for SC2.2 agents can be a set of scripts or programs. The agent for DB2 on Solaris is a set of programs that includes the following methods:

START_NET
hadb2_startnet, used to start DB2

STOP_NET
hadb2_stopnet, used to stop DB2

FM_START
hadb2_fmstart, used to start the fault monitor for DB2

FM_STOP
hadb2_fmstop, used to stop the fault monitor for DB2

For more information about these control methods, refer to the Sun Cluster documentation.

For EE instances, the logical host that is associated with the instance is defined right in the `hadb2tab` file. For EEE instances, however, the control method must also look in:

```
~<instance>/sqllib/ha/hadb2-eee.cfg
```

where `~<instance>` is the home directory of the instance owner. This file contains one line for each database partition, and is used to associate database partitions with logical hosts. An example of a valid `hadb2-eee.cfg` file is:

```
crackle % cat hadb2-eee.cfg
NODE:log0 0
NODE:log0 1
NODE:log1 2
NODE:log1 3
```

The instance or database partitions follow the corresponding logical host around the cluster. The logical host can move to any machine in the cluster that is supported by the underlying hardware and SC2.2. If the configuration is properly set up, DB2 will support any topology that is supported by the SC2.2 software.

After reading all of the information for an instance, the control method knows which logical hosts are associated with the instance. After parsing the command line arguments, the control method also knows which logical hosts are hosted, and which are not hosted by the current machine.

The following table shows the actions that are taken, depending on which control method is being run, and whether the logical hosts associated with the database partition or instance are hosted on the current machine.

Control Method	Associated logical host(s) are hosted	Associated logical host(s) are not hosted
START_NET	Start DB2 instance or database partitions	No action
STOP_NET	No action	Stop DB2 instance or database partitions
FM_START	Start fault monitor for instance	No action
FM_STOP	No action	Stop fault monitoring for instance

The control methods that perform start actions are only concerned with the logical hosts that are currently being hosted, and the control methods that perform stop actions are only concerned with the logical hosts that are not currently being hosted.

The control methods also need to mount the HA-NFS directory in a special way if HA-NFS is being used. If the local mount point and directory for HA-NFS are not defined as `-` (hyphen), the control method runs a `statvfs(2)` on the local mount point. If the file system type for the local mount point is not `nfs`, the agent attempts to mount the file system using information from the `hadb2tab` line. If the mount point and the directory for HA-NFS are defined as `-` (hyphen), the `vfstab` file of the corresponding logical host is required to mount the file system containing the home directory of the

instance. The local mount point and the remote directory for HA-NFS should only be defined as - (hyphen) for EE and EEE hot standby configurations.

User Scripts

These scripts are run from the control methods to add additional functionality; they are passed the same command line arguments as the control methods are passed, and are written by the system administrator or the database administrator.

If a program must be run from within a script that is not run in the background, consider backgrounding the program with `nohup(1)`. The `nohup` program protects the executed program from the SIGHUP (or hangup) signal. Without `nohup`, a program that is run in the background from a script may die as a result of a SIGHUP signal when the script is finished.

The control methods run the following scripts:

- `/var/db2/v61/failover`
- `~<instance>/sqllib/ha/pre_db2start`
- `~<instance>/sqllib/ha/post_db2start`
- `~<instance>%s/sqllib/ha/post_failover`
- `~<instance>/sqllib/ha/pre_db2stop`
- `~<instance>/sqllib/ha/fm_warning`

where *~instance* is the home directory of the HA instance.

With the exception of the `fm_warning` script, each user script is run with the same arguments as the control method that invoked it. When using EEE instances, the database partition number is also passed (as the last argument) to the user script.

The `/var/db2/v71/failover` script is invoked at the beginning of the START_NET method, and runs in the background. Such a script can be used, for example, to e-mail support staff in the event of a failover. Following is an example of a failover script:

```
#!/bin/ksh

# E-mail or page support staff to notify them that a failover has occurred.

echo "Failover occurred on machine 'hostname':Running $0!"
   |/bin/mail admin@sphere.torolab.ibm.com
```

To e-mail successfully from a script, `sendmail(1m)` must be properly configured on the system.

As its name suggests, the pre_db2start script is run just before **db2start** is invoked. This script can be used for such tasks as changing database manager configuration parameters. It is given a maximum of 20 seconds to complete. For EEE instances, this script is run before **db2start** is invoked on each database partition. This script is applicable only to data instances, not to administrative instances.

Similarly, the post_db2start script is run just *after* **db2start** is invoked. This script can be used for such tasks as restarting databases. It is run in the background to ensure that its execution time does not interfere with other instances. This script is applicable only to data instances, not to administrative instances.

The post_failover script under the instance owner's home directory, is run after processing the instance. This script can be used to notify client applications that DB2 is now functional, to activate databases, or to send administrators a status file. It is run in the background to prevent its execution time from delaying actions against the other HA instances. Following is an example of a post-failover script:

```
#!/bin/ksh
#

# Send the status file to the administrato-r.
mail admin@sphere.torolab.ibm.com </tmp/HA.info.db2eee
```

Both the START_NET and the STOP_NET method of the DB2 HA agent create a status file after processing each instance. The name of the status file is:

```
/tmp/HA.info.<instance>
```

where *instance* is the user name of the instance owner. The status file contains the start and stop report for the instance, as well as the time it took to run the control method. Following is an example of a status file:

```
scadmin@crackle(173)# cat /tmp/HA.info.db2eee
-----  Elapsed Time: 00:00:18           -----
-----  Elapsed Time: 00:00:00  (HA-NFS) -----

NODE     ACTION     RESULT     TRIES     RC
----     ------     ------     -----     --
   4     stop       success        3    1064
   5     stop       success        1    1064
   6     stop       success        2    1064
   7     stop       success        2    1064
   8     stop       success        1    1064
-------------------------------------------
```

The pre_db2stop script is run just before **db2stop** is invoked. This script can be used to notify client applications that DB2 is about to stop. It is given a

maximum of 20 seconds to complete. This script is applicable only to data instances, not to administrative instances.

The fault monitor will also run a user script when DB2 is restarted because of an unexpected shutdown. This script is called:

```
~<instance>/sqllib/ha/fm_warning
```

The `fm_warning` script can be used to notify the system administrator that DB2 was restarted by the fault monitor. The system administrator should try to find out why DB2 shut down unexpectedly, and take appropriate actions to prevent this from happening again. The `fm_warning` script is run in the background.

Other Considerations

If an HA data service is turned off, only the stop methods are run during a failover or cluster reconfiguration; the other methods are run only if the HA data service is properly registered and turned on.

Ensure that each machine in the cluster has enough resources to run all of the data services for which it may be responsible. Resources such as CPU load, memory, swap and kernel parameters must be considered before the cluster goes into production. For example, if a machine in the cluster may need to run two DB2 instances, the kernel parameter requirements for that machine will be the sum of what is needed for each instance.

Fault Monitor

If fault monitoring is turned on, the fault monitor will be started during a cluster reconfiguration or failover. If DB2 is not started by the START_NET script, the fault monitor itself will start DB2. The fault monitor can detect if DB2 did not start, or if it shut down for unknown reasons. Because of this, it is important not to shut down DB2 manually when the fault monitor is turned on. The fault monitor will see this as an unexpected shutdown, and restart DB2. If this happens too many times, it will fail over the appropriate logical host.

When fault monitoring is enabled for an instance, the correct way to start or stop the instance manually is to first turn off fault monitoring or the `hadb2` service. Both of these actions can be initiated through the **hadb2_setup** command using the `-f` and `-s` switches (see "The hadb2_setup Command" on page 281).

Note: Do not use more than one instance for the same logical host. If more than one instance is associated with a logical host, a healthy instance may be failed over along with an unhealthy one.

EEE Considerations

When deciding which database partitions to associate with a logical host, it is important to consider how they will fail over. Consider a two-machine cluster that is to be used with four database partitions between the two machines, as shown in Figure 34.

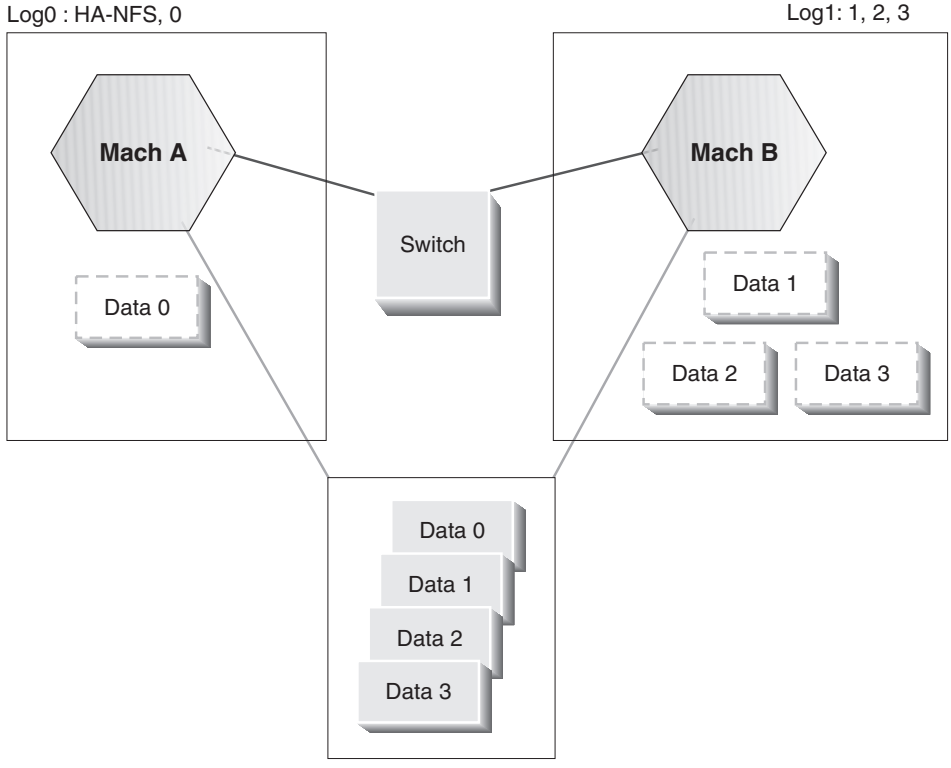


Figure 34. Two-machine Cluster with Four Database Partitions

You could associate one logical host with each database partition, and one for HA-NFS. In this case, there could be a problem if all of the logical hosts are being hosted by one system. If that system fails, all of the logical hosts must be moved off the system at the same time. Unfortunately, the Sun Cluster software does not move the logical hosts in any predictable order, and it is possible for a logical host that has a database partition associated with it to move before the logical host with HA-NFS. It is usually a good idea to group database partitions together, according to what would be hosted on a single system. This means that two database partitions that are normally hosted on one machine should be associated with a single logical host.

The db2nodes.cfg file used by an EEE instance is updated to indicate the machine on which the database partitions are residing. For example, if all of the database partitions are on a machine called "crackle", the db2nodes.cfg file resembles the following:

```
scadmin@crackle(193)# cat db2nodes.cfg
0 crackle 0 204.152.65.33
1 crackle 1 204.152.65.33
2 crackle 2 204.152.65.33
3 crackle 3 204.152.65.33
4 crackle 4 204.152.65.33
5 crackle 5 204.152.65.33
6 crackle 6 204.152.65.33
7 crackle 7 204.152.65.33
8 crackle 8 204.152.65.33
```

If some of these database partitions are moved to a machine called "thrash", the db2nodes.cfg file is updated as follows:

```
scadmin@crackle(193)# cat db2nodes.cfg
0 crackle 0 204.152.65.33
1 crackle 1 204.152.65.33
2 crackle 2 204.152.65.33
3 crackle 3 204.152.65.33
4 thrash 0 204.152.65.34
5 thrash 1 204.152.65.34
6 thrash 2 204.152.65.34
7 thrash 3 204.152.65.34
8 thrash 4 204.152.65.34
```

Notice that both the host name and the switch name are changed to reflect the machine name "thrash", and that the port numbers are also different.

The HA.config File

If it exists, the /etc/HA.config file can contain a number of configuration options, including the following:

```
scadmin@thrash(204)# cat /etc/HA.config
SYSLOG_FACILITY=LOG_LOCAL3
SYSLOG_LPRIORITY=LOG_INFO
SYSLOG_EPRIORITY=LOG_ERR
USE_INTERCONNECT=auto
SWITCH_NAME=204.152.65.18
DEBUG_LEVEL=2
FAILS_PER_HOUR=2
FAILS_PER_DAY=4
FAILS_PER_WEEK=10
FM_FAIL_SEV=soft
DB2START_TIMEOUT=60
DB2STOP_TIMEOUT=500
SCRIPT_USER=bin
```

Note: If the HA.config file does not exist, default values are used.

The SYSLOG_FACILITY variable sets the SYSLOG facility for logging both messages and errors. The SYSLOG_LPRIORITY and SYSLOG_EPRIORITY variables set the SYSLOG priority for logging informational messages and error messages, respectively.

Some changes may be needed to enable the SYSLOG daemon to log information from the DB2 HA agent. For example, one of the following two lines added to the `/etc/syslog.conf` file will tell the SYSLOG daemon to write information to a log file.

```
*.notice                                        /var/adm/SC.x
local3.info                                     /var/adm/SC.LOG_LOCAL3
```

A Sun Cluster usually has a high speed interconnect. To use the high speed interconnect with DB2, set USE_INTERCONNECT to `auto` or to `override`. The `auto` setting (the default) uses the Sun internal logical network interface. This interface will be transferred to another physical interface if the initial interface fails. If USE_INTERCONNECT is set to `override`, the switch name is taken from the SWITCH_NAME variable. Another option is to set USE_INTERCONNECT to `no`, which specifies that high speed interconnect is not to be used.

DEBUG_LEVEL specifies how much information is to be logged during a failover. It is a number between 0 and 10, where 10 is the highest debug level. The information is logged at the specified SYSLOG priority and facility. If any problems are encountered, set the debug level to the maximum level, configure SYSLOG to log the output from the HA agents, and send the SYSLOG output to IBM service.

Three of the variables help the DB2 fault monitor decide when to fail over a logical host: FAILS_PER_HOUR, FAILS_PER_DAY, and FAILS_PER_WEEK. Every HA environment is different; you must decide how many DB2 failures are acceptable. After each ″acceptable″ failure, DB2 is restarted on the same machine. When one of these three failure thresholds is exceeded, the logical host associated with the instance or database partition is failed over.

The FM_FAIL_SEV variable specifies whether the failover is ″soft″ or ″hard″. For more information, refer to the Sun Cluster documentation on `hactl(1m)`.

The DB2START_TIMEOUT and DB2STOP_TIMEOUT variables specify the maximum number of seconds that **db2start** and **db2stop** are allowed to run. After the specified interval has passed, the HA agent considers the operation to have failed, and try to restart the instance.

There are some user scripts that are not associated with any particular instance. Normally, these scripts are run as root; this can be overridden by the SCRIPT_USER variable, which can be set to specify the user ID that can run these scripts.

How Control Methods Run DB2 Commands

The DB2 HA agent uses the **su** command to run commands as the instance owner. The actual command would look something like:

```
su - <instance> -c "db2stop"
```

where *instance* is the user name of the instance.

It is important to ensure that the `.profile` file of the instance owner is **su**-"friendly". If it is not, the **su** command may not work properly. Invoke the **su** command manually, or from a script, to verify that the command can run successfully.

Setup

Before you read this section, be sure that you are familiar with the SC2.2 software. This section assumes that you know how to set up SC2.2 and HA-NFS, and that you know how to use your volume manager. Along with the other required patches for DB2, the following patches are required for the HA agent:

```
Solaris 2.6:
   105210-17 (or later)
   105786-05 (or later)
```

Note: There are no required patches for Solaris 7 (Solaris 2.7).

Common Installation Steps

1. Install SC2.2 on all machines in the cluster. During installation, SC2.2 will ask which agents to install. Since DB2 is not shipped with SC2.2, it is not in the list of agents. The agent for DB2 will be installed with DB2 and registered through the **hadb2_reg** command.
2. Configure the logical hosts with disk groups and logical IP addresses.

Setup on DB2 UDB Enterprise Edition

1. Create the home directory for the instance under the logical host file system of a logical host.
2. Install DB2 on all machines in the cluster.
3. Create the instance on the machine in the cluster that currently has the home directory for the instance.
4. Add the user for the instance to the other machines in the cluster, ensuring that the numeric user ID is the same.

5. Register the hadb2 service using the **hadb2_reg** command.
6. Run the **hadb2_setup** command to set up HA for the instance.

Setup on DB2 UDB Enterprise - Extended Edition

1. Create the home directory for the HA instance owner:
   a. For hot standby, create the home directory for the instance under the logical host file system of a logical host.
   b. For mutual takeover, configure HA-NFS, and export the home directory from one of the logical hosts. On one of the machines, mount the HA-NFS directory under the chosen mount point.
2. Install DB2 on all machines in the cluster.
3. Create the instance on the machine that has the HA-NFS file system mounted.
4. Add the user for the instance to the other machines in the cluster, ensuring that the numeric user ID is the same.
5. Register the hadb2 service using the **hadb2_reg** command.
6. Run the **hadb2_setup** command to set up HA for the instance.

Note: Using NIS to define the information for the HA instance is not recommended, because NIS can introduce a single point of failure.

The hadb2_setup Command

The **hadb2_setup** command is the central point of the programs that come with the DB2 HA agent. It can be used to set up an instance, to modify it, or to delete it. It can also be used to turn the hadb2_setup service on and off. With this command, there is no need to manually edit the hadb2tab file.

Note: The **hadb2_setup** command performs actions only on the machine on which it runs. Changes made to one machine should also be made to the other machines in the cluster.

The following arguments are supported:

```
To add an EE instance:
----------------------
   hadb2_setup -a -i <instance> -f [on|off] -h <logical_host> -p [DATA|ADMIN] -t EE

   For example:
      hadb2_setup -a -i db2ee -f off -h log1 -p DATA -t EE



To add an EEE instance:
-----------------------
   hadb2_setup -a -i <instance> -f [on|off] -h <nfs_host> -l <mount_point> \
      -r <ha-nfs_dir> -p [DATA|ADMIN] -t EEE -n "<node_info>"

   For example:
```

```
      hadb2_setup -a -i db2eee -f off -h ha-sun1 -l /export/ha_home \
         -r /log0/home -p DATA -t EEE -n "log0[0,10,20],log1[30,40,50]"


To delete an instance:
----------------------
   hadb2_setup -d -i <instance>


To modify an instance:
----------------------
   hadb2_setup -m -i <instance> [-f [on|off] | -l <mount_point> | \
      -h <host> | -p [DATA|ADMIN] | -r <ha-nfs_dir> | -t [EE|EEE] ]


Other options:
--------------
   -s   <on|off>             Bring hadb2 up or down (for all HA instances)
   -y                        Assume yes for safety checks
```

To turn the hadb2 service on or off, specify the -s switch. This is equivalent to using the **hareg** command with the -n and -y switches, and specifying the hadb2 service. For more information about the **hareg(1m)** command, refer to the Sun Cluster documentation.

The fault monitor for the instance can be turned off using the -f switch. This has the effect of stopping the fault monitor for the instance on the local machine, as well as modifying the hadb2tab file to reflect the fact that fault monitoring is turned off.

For EE instances, turning off fault monitoring on all machines is recommended in case the instance fails over. For EEE instances, fault monitoring must be turned off on all machines that are hosting database partitions for the instance before it is shut down manually.

To delete an instance, use the -d switch. This only removes the instance from the hadb2tab file, and does not remove or modify any other files or directories. Since the hadb2tab file is the main configuration file for the HA-DB2 agent, removing an instance from this file makes the control methods unaware of its existence.

To modify an instance, use the -m switch. This only changes information in the hadb2tab file, and does not remove or modify any other files or directories. The -m switch can be used with any switch that pertains to information in the hadb2tab file. The db2nodes.cfg file and the hadb2-eee.cfg file must be changed manually after the initial setup, because the **hadb2_setup** command does not support modifying these files.

Adding an instance is somewhat more involved.

For EE instances, the following arguments are required:

```
hadb2_setup -a -i <instance> -f <fm> -h <logical_host> -t <EEE_or_EE>
   -p <purpose>
```

where *instance* is the name of the instance to be added, *fm* specifies whether fault monitoring is initially turned on or off, *logical_host* is the associated logical host, *EEE_or_EE* is set to `EE`, and *purpose* can be either `DATA` or `ADMIN`.

For EEE instances, the following arguments are required:

```
hadb2_setup -a -i <instance> -f <fm> -h <nfs_host> -t <EEE_or_EE> -p
   <purpose> -l <mount_point> -r <HA-NFS_directory> -n <node_info>
```

where *instance* is the name of the instance to be added, *fm* specifies whether fault monitoring is initially turned on or off, *nfs_host* is the host name for the logical host that is exporting the HA-NFS file system, *EEE_or_EE* is set to `EEE`, *purpose* can be either `DATA` or `ADMIN`, *mount_point* is the local mount point for the HA-NFS directory, *HA-NFS_directory* is the HA-NFS directory, and *node_info* is the information that associates database partitions with a logical host. For example:

```
hadb2_setup -a -i db2eee -f on -h jolt -l /export/ha_home -p DATA -t EEE -r
   /log1/home -n "log0[0,1],log1[2,3]"
```

When adding an EEE instance, the node information must be enclosed by quotation marks. In this example, the instance ″db2eee″ will be associated with two logical hosts, ″log0″ and ″log1″. Database partitions ″0″ and ″1″ of the ″db2eee″ instance will be associated with the logical host ″log0″, and database partitions ″2″ and ″3″ will be associated with logical host ″log1″.

Use the **hadb2_setup** command to add an instance to all machines in the cluster. The instance can then be started by forcing a cluster reconfiguration, or by turning `hadb2` service off and then on. This can be done, either through the **hareg** command, or with the `-s` switch of the **hadb2_setup** command. If the instance does not start, see “Troubleshooting” on page 287.

When the **hadb2_setup** command adds an EEE instance, the following actions are performed transparently:

- Checking the specified information. This includes ensuring that the user exists on the system, and that HA-NFS is running.
- Creating a `db2nodes.cfg` file.
- Creating an `hadb2-eee.cfg` file.
- Creating a `.rhosts` file for the EEE instance.
- Creating symbolic links from the default database path to the associated logical hosts data directories.

- Adding a line to the hadb2tab file.

To prevent configuration errors, and to ensure that the HA instance will be able to start after the **hadb2_setup** command runs, the command performs a significant amount of testing before a new instance is added.

The db2nodes.cfg file is created and seeded with information corresponding to the current cluster status. For example, if logical host "log0" is being hosted by the machine "crackle", the entries for the database partitions associated with "log0" will contain the machine name "crackle" and the high speed interconnect for "crackle":

```
scadmin@crackle(193)# cat db2nodes.cfg
0 crackle 0 204.152.65.33
1 crackle 1 204.152.65.33
2 thrash 0 204.152.65.34
3 thrash 1 204.152.65.34
```

The hadb2-eee.cfg file is created only on the basis of the node information that is specified on the command. There is one line per database partition:

```
sphere % cat hadb2-eee.cfg
NODE:log0 0
NODE:log0 1
NODE:log1 2
NODE:log1 3
```

The .rhost file is required for DB2 UDB EEE, and should contain all host names (or IP addresses) for each machine in the cluster. For example:

```
crackle db2eee
204.152.65.1 db2eee
204.152.65.17 db2eee
thrash db2eee
204.152.65.2 db2eee
204.152.65.18 db2eee
crackle db2eee
jolt db2eee
bump db2eee
thrash.torolab.ibm.com db2eee
crackle.torolab.ibm.com db2eee
```

In accordance with a file system layout for SMS tables spaces, the **hadb2_setup** command sets up a number of directories and symbolic links. These include:

- A directory called "data" under the logical host file system for each logical host.
- A node directory (under this "data" directory) for each database partition associated with the logical host.
- Symbolic links in the default database path, located under ~<instance>, where *~instance* is the home directory of the instance. There is one symbolic

link for each database partition that points to the corresponding node directory. For more information, see "Disk Layout for EE and EEE Instances" on page 266.

Failover Time

Failover time is measured from when data is first unavailable to when it is available again. A number of events that occur during a failover can contribute significantly to the failover time:

- Disk deporting and importing.

  Deporting and importing disks usually does not take a very long time compared to other events, although it does contribute to the overall down time. The more disks that need to be moved from one machine to another during a failover, the longer the process takes. If there are defective disks, the process can take even longer.

- Fsck of the file systems that are mounted for a logical host.

  Before the file systems of the logical host can be mounted, they must pass an `fsck` to ensure the health of the file system. The larger the file system, the longer this process takes. By using a journalled file system, this time can be drastically reduced. Since journalled file systems are normally used in an HA environment, the `fsck` time is usually not an issue.

- User scripts called from the HA agent.

  The HA agent will call user scripts if they exist and are executable. Some of these scripts are run synchronously, and can add to the time it takes to bring up the HA instances. Ensure that they run as quickly as possible; consider running any external programs called by these scripts in the background.

- HA-NFS.

  For a single EEE instance in a mutual takeover configuration, HA-NFS must be used for the home directory of the instance owner. HA-NFS adds to failover time because of the grace period for *lockd* (defined in the HA agent for HA-NFS), which is 90 seconds when running HA-NFS. This affects failover times, because any process that locks a file on the HA-NFS file system after a failover must wait until the grace period is over. The HA agent for DB2 is the first process to lock a file under the instance owner's home directory after a failover, and it records the time it takes to obtain the first lock. This time is displayed in the status report after a failover.

- Starting DB2.

  Starting DB2 contributes only a small amount to the failover time. For an EE instance, it contributes about 5-15 seconds on average. For an EEE instance, it contributes about 10 seconds, plus about 5 seconds per database partition that is being failed over. If three database partitions are being failed over, for example, the failover time contributed by starting these

three database partitions will be approximately 25 seconds. This does *not* include crash recovery for the databases of the instance.

- Database crash recovery.

  Crash recovery often contributes to the majority of down time associated with a failover. How long it takes to recover a database depends on a number of factors, including:

  – Client workload. Only changes to the database are logged in the transaction logs. If the client workload is mostly read-only operations, relatively few transactions must be applied to the database during crash recovery.

  – Disk and machine speed. The speed of the disks and the machine that is hosting the HA instance also contributes to the time it takes to recover the database. The faster the system, the shorter the crash recovery time.

  – Value of the *softmax* database configuration parameter. The value of *softmax* is the percentage of the log file size at which a soft checkpoint is to be taken, and a log control file is to be written. The log control file is used during crash recovery to determine which log records are truly necessary to restore the database to a consistent state. Reducing this value will cause the database manager to trigger the page cleaners more often, and take more frequent soft checkpoints; although performance is reduced, database recovery is faster.

  – Whether the instance is EE or EEE. If the instance is an EEE instance, the database restart operations will be done in parallel. Each database partition is responsible for restarting its own portion of the databases. If there are 50 GB of data for a database, an instance with four database partitions will be able to recover the database roughly four times faster than an EE instance can.

Troubleshooting

The following table identifies problems that you might encounter, their probable causes, and actions that you can take to solve them.

Table 16. Troubleshooting High Availability on Sun Cluster 2.2

Symptom	Possible cause	Action
Cannot mount logical host file system	The logical host file system is normally mounted and unmounted during the failover of a logical host. During failover, there should be no active processes or open files under the logical host file system. In rare cases, processes that cannot be killed have their current working directory under the logical host file system. To find out if a process is under the mount point, use `fuser(1m)`, or a GNU utility called `lsof`. Error messages are produced when the logical host file system cannot be mounted.[a]	Reboot the system, or move the logical host file system to another name and recreate it. Doing this allows the frozen process to stay under the directory (since it can't be killed), and allows the mount to take place.[b]
The db2start or db2stop time-out does not work	A SIGALRM signal may not break out of a blocking system call. Instead, the system call will restart as if the SA_RESTART flag were set with `sigaction()`. This causes time-outs for the DB2 HA agents to be ignored, and the agent method will hang instead of recovering from a hung **db2start** or **db2stop** command.	Apply the required patch, 105210-17 (or later), for Solaris 2.6.
Logging into an instance hangs	Although there are numerous reasons why this can happen, the most common reasons include NFS problems and the `/usr/sbin/quota` program.	Check the NFS mounts to ensure that they are healthy, and look for quota processes owned by the instance owner. At the discretion of the system administrator, changing the quota program to a symbolic link to `/bin/true` may solve the problem. This is not a recommended solution, but it may work.
I just set up an EEE instance, but it does not start	The **hadb2_setup** command does not add ports to the `/etc/services` file; it is expected that the administrator will add them manually. An error message is returned.[c]	Ensure that you have appropriate ports named in the `/etc/services` file.

Table 16. Troubleshooting High Availability on Sun Cluster 2.2 (continued)

Symptom	Possible cause	Action
START_NET method cannot start DB2		Turn off fault monitoring to ensure that the instance does not get failed over. Log in as the instance owner, and try to start DB2 manually. 1. Ensure that the `hadb2tab` configuration file has the correct instance type specified. For example, having a `db2nodes.cfg` file for an EE administrative instance will cause problems, and the HA agent methods will not be able to recover from this. 2. Ensure that the `.rhosts` file exists, and has valid entries in it. 3. Ensure that the HA-NFS file system is shared with root permissions for all machines in the cluster. 4. Check the kernel parameters, and ensure that they are correct. 5. Ensure that the `/etc/services` file contains entries for the instance.
The instance only works on one machine	• The numeric *uid* for the instance may not be the same on each machine in the cluster. • The kernel parameters may not be valid on each machine in the cluster. • The `hadb2tab` file may not be the same on each machine in the cluster. • Other configuration files, such as the logical host `vfstab` file, may not be the same on each machine in the cluster.	If none of these causes appears to apply, try logging in as the instance owner, and start DB2 manually. For EE instances, this should work if the logical host that is hosting the instance is being hosted by the current machine. For EEE instances, this should work from any machine in the cluster that can host the database partitions.

Table 16. Troubleshooting High Availability on Sun Cluster 2.2 (continued)

Symptom	Possible cause	Action
su - <instance> -c ″db2start″ does not work	• The `.profile` for the instance may not be **su**-″friendly″. • There is a known problem with the Bourne shell (`/bin/sh`), in which the **su** command works manually, but not through the HA agent.	• As root, try running this command manually, and ensure that it works before trying again through the HA agent. • Switch to the Korn shell (`/bin/ksh`), if necessary.
My EEE instance cannot start, but the home directory is mounted	The HA-NFS directory may not have been exported with ″root″ permissions to the machines in the cluster. Both DB2 and the HA agents require this to run properly.	To test this, try to create a file (as root) under the instance owner's home directory.
Trying to access the EEE instance directory returns a ″Stale NFS file handle″ error	There may still be processes under the instance owner's home directory.	Unmount the instance owner's home directory, and allow the HA agent to remount it. The HA agent will remount it if the `hadb2` service is turned off and on again (see a description of the `-s` switch on the **hadb2_setup** command in "The hadb2_setup Command" on page 281).

Table 16. Troubleshooting High Availability on Sun Cluster 2.2 (continued)

Symptom	Possible cause	Action
Control methods do not run successfully through SC2.2	The `hadb2` service may not be registered with the Sun Cluster software, or it may not be turned on.	If the control methods appear to run normally from the command line, check the SYSLOG files for error messages that may help to explain the problem. Ensure that the `hadb2` service is registered with the Sun Cluster software, and that it is turned on. Running the methods manually is useful for debugging a problem.[d] The methods should be run as root and given the appropriate command line arguments. If the list of logical hosts is nil, the argument should be given as `""`. The double quotation marks without a blank space separator denotes a blank argument. For example: `hadb2_startnet log0,log1 "" 600` The first argument, `log0,log1`, tells the `hadb2_startnet` method that logical hosts `log0` and `log1` are being hosted by the current machine. The second argument is nil, which tells the `hadb2_startnet` method that there are no other logical hosts being hosted on other machines in the cluster (all of them are on the current machine). The third argument tells the method that SC2.2 will time out after 600 seconds.
User scripts do not run	The user scripts can only be run if they exist in the appropriate directories and are executable.	Check file ownership and attributes. If a script still fails to run, contact IBM service. Forward a directory listing of the script that does not run, and SYSLOG output for a failover or a cluster reconfiguration that should have run the script.

Table 16. Troubleshooting High Availability on Sun Cluster 2.2 (continued)

Symptom	Possible cause	Action
Information is not being logged to the file specified in /etc/syslog.conf		Use `touch(1)` to create the file that is specified in the `/etc/syslog.conf` file, and then restart the SYSLOG daemon.

[a] Error messages that are produced when the logical host file system cannot be mounted may look something like the following:

```
Aug 17 11:14:01 rash ID[SUNWcluster.loghost.1170]: importing data1
Aug 17 11:14:06 rash ID[SUNWcluster.scnfs.3040]: mount -F ufs -o ""
     /dev/vx/dsk/data1/data1-stat /log1 failed.
Aug 17 11:14:07 rash ID[SUNWcluster.ccd.ccdd.5304]: error freeze cmd =
     /opt/SUNWcluster/bin/loghost_sync
CCDSYNC_POST_ADDU LOGHOST_CM:log1:rash /etc/opt/SUNWcluster/conf/ccd.database
     2 "0 1" 1 error code = 1
```

[b] For example:

```
scadmin@rash(218)# ps -fe | egrep db2
db2ee 1984 1 0 0:01 <defunct>

Solution:

   scadmin@rash(229)# cd /
   scadmin@rash(230)# mv /log1 /log1.bkp
   scadmin@rash(231)# mkdir /log1
```

[c] The error message may look something like the following:

```
SQL6030N START or STOP DATABASE MANAGER failed. Reason code "13".
```

[d] For example, if the `hadb2_startnet` method cannot find `libdb2.so.1`, but it runs normally through the Sun Cluster software, no errors will be reported. Running the method manually results in the following:

```
scadmin@crackle(213)# hadb2_startnet '''log0,log1' 600
ld.so.1: hadb2_startnet: fatal: libdb2.so.1: open failed:
   No such file or directory
Killed
```

Part 3. Appendixes

Appendix A. How to Read the Syntax Diagrams

A syntax diagram shows how a command should be specified so that the operating system can correctly interpret what is typed.

Read a syntax diagram from left to right, and from top to bottom, following the horizontal line (the main path). If the line ends with an arrowhead, the command syntax is continued, and the next line starts with an arrowhead. A vertical bar marks the end of the command syntax.

When typing information from a syntax diagram, be sure to include punctuation, such as quotation marks and equal signs.

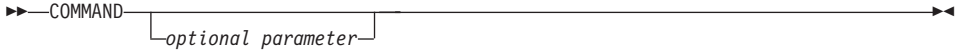
Parameters are classified as keywords or variables:

- Keywords represent constants, and are shown in uppercase letters; at the command prompt, however, keywords can be entered in upper, lower, or mixed case. A command name is an example of a keyword.
- Variables represent names or values that are supplied by the user, and are shown in lowercase letters; at the command prompt, however, variables can be entered in upper, lower, or mixed case, unless case restrictions are explicitly stated. A file name is an example of a variable.

A parameter can be a combination of a keyword and a variable.

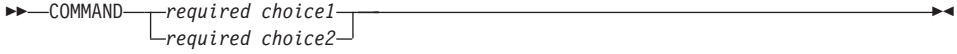
Required parameters are displayed on the main path:

Optional parameters are displayed below the main path:



A parameter's default value is displayed above the path:

A stack of parameters, with the first parameter displayed on the main path, indicates that one of the parameters must be selected:



A stack of parameters, with the first parameter displayed below the main path, indicates that one of the parameters can be selected:

An arrow returning to the left, above the path, indicates that items can be repeated in accordance with the following conventions:

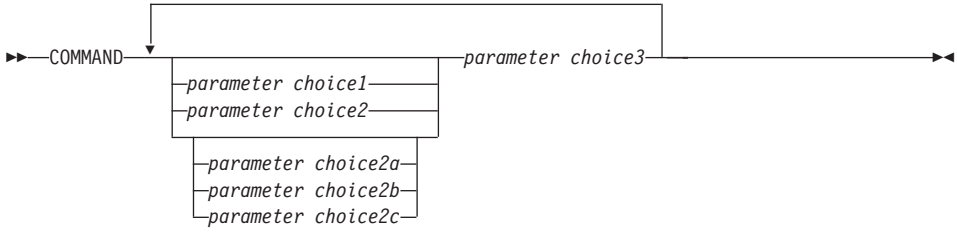
- If the arrow is uninterrupted, the item can be repeated in a list with the items separated by blank spaces:

- If the arrow contains a comma, the item can be repeated in a list with the items separated by commas:

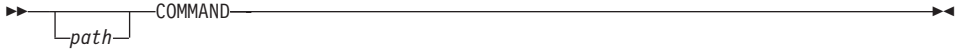
Items from parameter stacks can be repeated in accordance with the stack conventions for required and optional parameters discussed previously.

Some syntax diagrams contain parameter stacks within other parameter stacks. Items from stacks can only be repeated in accordance with the

conventions discussed previously. That is, if an inner stack does not have a repeat arrow above it, but an outer stack does, only one parameter from the inner stack can be chosen and combined with any parameter from the outer stack, and that combination can be repeated. For example, the following diagram shows that one could combine parameter *choice2a* with parameter *choice2*, and then repeat that combination again (*choice2* plus *choice2a*):

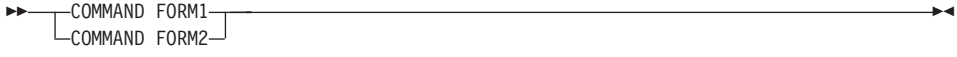


Some commands are preceded by an optional path parameter:



If this parameter is not supplied, the system searches the current directory for the command. If it cannot find the command, the system continues searching for the command in all the directories on the paths listed in the `.profile`.

Some commands have syntactical variants that are functionally equivalent:



Appendix B. Warning, Error, and Completion Messages

Messages generated by the various backup and recovery utilities are included among the SQL messages. These messages are generated by the database manager when a warning or error condition has been detected. Each message has a message identifier that consists of a prefix (SQL) and a four- or five-digit message number. There are three message types: notification, warning, and critical. Message identifiers ending with an `N` are error messages. Those ending with a `W` indicate warning or informational messages. Message identifiers ending with a `C` indicate critical system errors.

The message number is also referred to as the *SQLCODE*. The SQLCODE is passed to the application as a positive or negative number, depending on its message type (N, W, or C). N and C yield negative values, whereas W yields a positive value. DB2 returns the SQLCODE to the application, and the application can get the message associated with the SQLCODE. DB2 also returns an *SQLSTATE* value for conditions that could be the result of an SQL statement. Some SQLCODE values have associated SQLSTATE values.

For detailed information about all of the DB2 messages, see the *Message Reference*. You can use the information contained in this book to identify an error or problem, and to resolve the problem by using the appropriate recovery action. This information can also be used to understand where messages are generated and logged.

SQL messages, and the message text associated with SQLSTATE values, are also accessible from the operating system command line. To access help for these error messages, enter the following at the operating system command prompt:

```
db2 ? SQLnnnnn
```

where *nnnnn* represents the message number. On UNIX based systems, the use of double quotation mark delimiters is recommended; this will avoid problems if there are single character file names in the directory:

```
db2 "? SQLnnnnn"
```

The message identifier accepted as a parameter for the **db2** command is not case sensitive, and the terminating letter is not required. Therefore, the following commands will produce the same result:

```
db2 ? SQL0000N
db2 ? sql0000
db2 ? SQL0000n
```

If the message text is too long for your screen, use the following command (on UNIX based operating systems and others that support the ″more″ pipe):

```
db2 ? SQLnnnnn | more
```

You can also redirect the output to a file which can then be browsed.

Help can also be invoked from interactive input mode. To access this mode, enter the following at the operating system command prompt:

```
db2
```

To get DB2 message help in this mode, type the following at the command prompt (`db2 =>`):

```
? SQLnnnnn
```

The message text associated with SQLSTATEs can be retrieved by issuing:

```
db2 ? nnnnn
 or
db2 ? nn
```

where *nnnnn* is a five-character SQLSTATE value (alphanumeric), and *nn* is a two-digit SQLSTATE class code (the first two digits of the SQLSTATE value).

Appendix C. Additional DB2 Commands

db2adutl - Work with TSM Archived Images

Allows users to query, extract, verify, and delete backup images, logs, and load copy images saved using Tivoli Storage Manager (formerly ADSM).

On UNIX based systems, this utility is located in the INSTHOME/sqllib/misc directory. On Windows operating systems and OS/2, it is located in the \sqllib\misc directory.

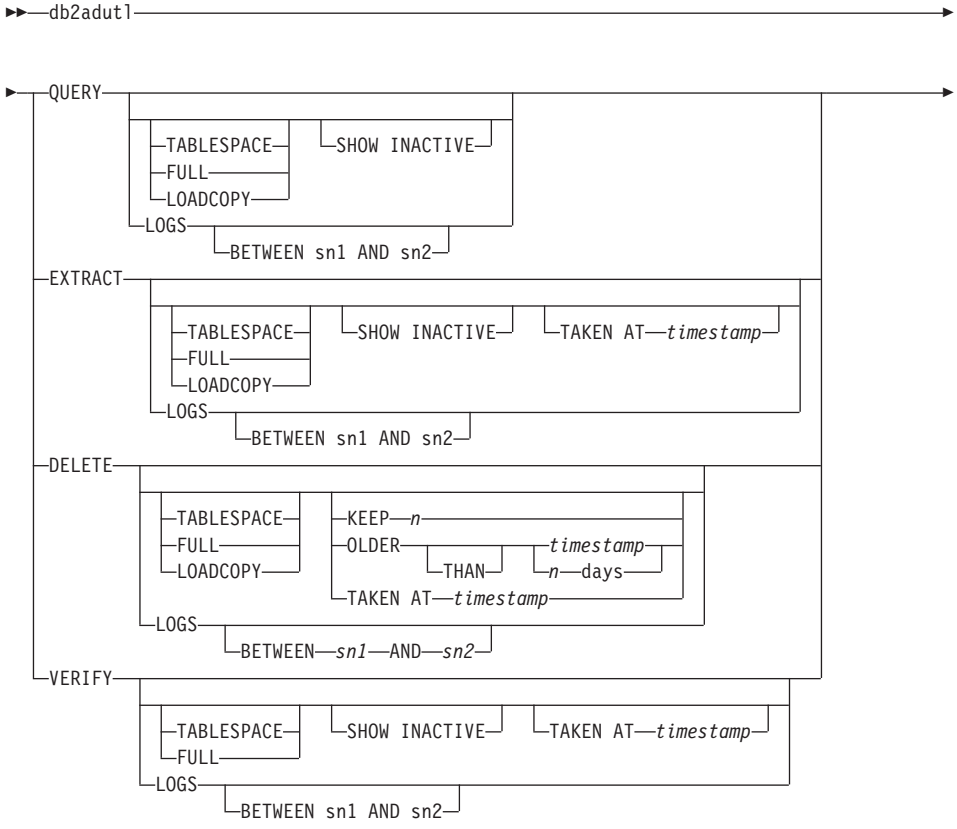
Authorization

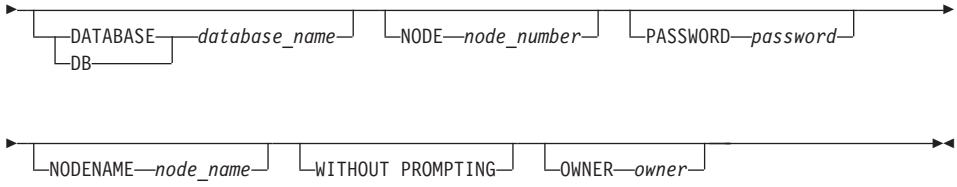
None

Required Connection

None

Command Syntax





Command Parameters

QUERY
Queries the TSM server for DB2 objects.

EXTRACT
Copies DB2 objects from the TSM server to the current directory on the local machine.

DELETE
Either deactivates backup objects or deletes log archives on the TSM server.

VERIFY
Performs consistency checking on the backup copy that is on the server.

Note: This parameter causes the entire backup image to be transferred over the network.

TABLESPACE
Includes only table space backup images.

FULL Includes only full database backup images.

LOADCOPY
Includes only load copy images.

LOGS Includes only log archive images

BETWEEN *sn1* **AND** *sn2*
Specifies that the logs between log sequence number 1 and log sequence number 2 are to be used.

SHOW INACTIVE
Includes backup objects that have been deactivated.

TAKEN AT *timestamp*
Specifies a backup image by its time stamp.

KEEP *n*
Deactivates all objects of the specified type except for the most recent *n* by time stamp.

OLDER THAN *timestamp* **or** *n* **days**
Specifies that objects with a time stamp earlier than *timestamp* or *n* days will be deactivated.

DATABASE *database_name*
Considers only those objects associated with the specified database name.

NODE *node_number*
Considers only those objects created by the specified node number.

PASSWORD *password*
Specifies the TSM client password for this node, if required. If a database is specified and the password is not provided, the value specified for the *tsm_password* database configuration parameter is passed to TSM; otherwise, no password is used.

NODENAME *node_name*
Considers only those images associated with a specific TSM node name.

WITHOUT PROMPTING
The user is not prompted for verification before objects are deleted.

OWNER *owner*
Considers only those objects created by the specified owner.

Examples

The following is sample output from: `db2 backup database rawsampl use tsm`

```
Backup successful. The timestamp for this backup is : 19970929130942

db2adutl query

Query for database RAWSAMPL

Retrieving full database backup information.
   full database backup image: 1, Time: 19970929130942,
                                  Oldest log: S0000053.LOG, Sessions used: 1
   full database backup image: 2, Time: 19970929142241,
                                  Oldest log: S0000054.LOG, Sessions used: 1

Retrieving table space backup information.
   table space backup image: 1, Time: 19970929094003,
                                Oldest log: S0000051.LOG, Sessions used: 1
   table space backup image: 2, Time: 19970929093043,
                                Oldest log: S0000050.LOG, Sessions used: 1
   table space backup image: 3, Time: 19970929105905,
                                Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
   Log file: S0000051.LOG
```

```
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

The following is sample output from: `db2adutl delete full taken at 19950929130942 db rawsampl`

```
  Query for database RAWSAMPL

  Retrieving full database backup information.  Please wait.

    full database backup image: RAWSAMPL.0.db26000.0.19970929130942.001

    Do you want to deactivate this backup image (Y/N)? y

      Are you sure (Y/N)? y

  db2adutl query

  Query for database RAWSAMPL

  Retrieving full database backup information.
     full database backup image: 2, Time: 19950929142241,
                               Oldest log: S0000054.LOG, Sessions used: 1

  Retrieving table space backup information.
     table space backup image: 1, Time: 19950929094003,
                             Oldest log: S0000051.LOG, Sessions used: 1
     table space backup image: 2, Time: 19950929093043,
                             Oldest log: S0000050.LOG, Sessions used: 1
     table space backup image: 3, Time: 19950929105905,
                             Oldest log: S0000052.LOG, Sessions used: 1

  Retrieving log archive information.
     Log file: S0000050.LOG
     Log file: S0000051.LOG
     Log file: S0000052.LOG
     Log file: S0000053.LOG
     Log file: S0000054.LOG
     Log file: S0000055.LOG
```

db2ckbkp - Check Backup

This utility can be used to test the integrity of a backup image and to determine whether or not the image can be restored. It can also be used to display the meta-data stored in the backup header.

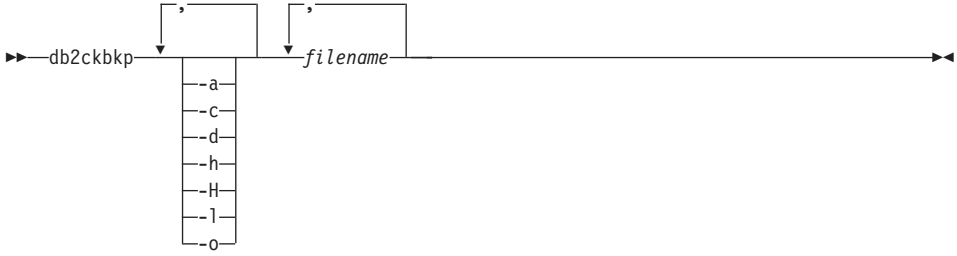
Authorization

Anyone can access the utility, but users must have read permissions on image backups in order to execute this utility against them.

Required Connection

None

Command Syntax



Command Parameters

-a Displays all available information.

-c Displays results of checkbits and checksums.

-d Displays information from the headers of DMS table space data pages.

-h Displays media header information, including the name or path of the image expected by the restore utility.

-H Displays only the media header information.

Notes:

1. This option does not validate the image. Validation is performed on the entire image if this option is not specified.
2. This option is not valid in combination with any other option.

-l Displays Log File Header data.

-o Displays detailed information from the object headers.

filename

The name of the backup image file. One or more files can be checked at a time.

Notes:

1. If the complete backup consists of multiple objects, the validation will only succeed if **db2ckbkp** is used to validate all of the objects at the same time.
2. When checking multiple parts of an image, the first backup image object (.001) must be specified first.

Examples

```
db2ckbkp SAMPLE.0.krodger.NODE0000.CATN0000.19990817150714.*
[1] Buffers processed:  ##
[2] Buffers processed:  ##
[3] Buffers processed:  ##
Image Verification Complete - successful.

db2ckbkp -h SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001


=====================
MEDIA HEADER REACHED:
=====================
        Server Database Name           -- SAMPLE2
        Server Database Alias          -- SAMPLE2
        Client Database Alias          -- SAMPLE2
        Timestamp                      -- 19990818122909
        Node                           -- 0
        Instance                       -- krodger
        Sequence Number                -- 1
        Release ID                     -- 900
        Database Seed                  -- 65E0B395
        DB Comment's Codepage (Volume) -- 0
        DB Comment (Volume)            --
        DB Comment's Codepage (System) -- 0
        DB Comment (System)            --
        Authentication Value           -- 255
        Backup Mode                    -- 0
        Backup Type                    -- 0
        Backup Gran.                   -- 0
        Status Flags                   -- 11
        System Cats inc                -- 1
        Catalog Node Number            -- 0
        DB Codeset                     -- ISO8859-1
        DB Territory                   --
        Backup Buffer Size             -- 4194304
        Number of Sessions             -- 1
        Platform                       -- 0

The proper image file name would be:
SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001

[1] Buffers processed:  ####
Image Verification Complete - successful.
```

Usage Notes

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.

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 NT:

```
db2ckbkp -d \\.\tape1
```

If the backup image resides on TSM, see "db2adutl - Work with TSM Archived Images" on page 302.

db2ckrst - Check Incremental Restore Image Sequence

Queries the database history and generates a list of time stamps for the backup images required for an incremental restore operation. A simplified restore syntax for a manual incremental restore is also generated.

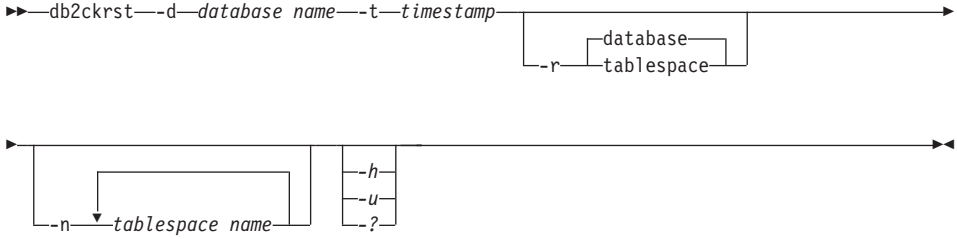
Authorization

None

Required Connection

None

Command Syntax



Command Parameters

-d database namefile-name
Specifies the alias name for the database that will be restored.

-t timestamp
Specifies the time stamp for a backup image that will be incrementally restored.

-r Specifies the type of restore that will be performed. The default is database.

Note: If table space is chosen and no table space names are given, the utility looks into the history entry of the specified image and uses the table space names listed to do the restore.

-n tablespace name
Specifies the name of one or more table spaces that will be restored.

Note: If the database restore type is selected and a list of table space names is specified, the utility will continue with a table space restore operation using the specified table space names.

-h/-u/-?
Displays help information. When this option is specified, all other options are ignored, and only the help information is displayed.

Examples

```
db2ckrst -d mr -t 20001015193455 -r database
db2ckrst -d mr -t 20001015193455 -r tablespace
db2ckrst -d mr -t 20001015193455 -r tablespace -n tbsp1 tbsp2

> db2 backup db mr

Backup successful. The timestamp for this backup image is : 20001016001426

> db2 backup db mr incremental

Backup successful. The timestamp for this backup image is : 20001016001445

> db2ckrst -d mr -t 20001016001445

Suggested restore order of images using timestamp 20001016001445 for database mr.
==============================================================
  db2 restore db mr incremental taken at 20001016001445
  db2 restore db mr incremental taken at 20001016001426
  db2 restore db mr incremental taken at 20001016001445
==============================================================

> db2ckrst -d mr -t 20001016001445 -r tablespace -n userspace1
Suggested restore order of images using timestamp 20001016001445 for database mr.
==============================================================
  db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at 20001016001445
  db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at 20001016001426
  db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at 20001016001445
==============================================================
```

Usage Notes

The database history must exist in order for this utility to be used. If a database history does not exist, specify the HISTORY FILE option in the RESTORE DATABASE command before using this utility.

If the FORCE option of the PRUNE HISTORY command is used, it will be possible to delete entries that are required for recovery from the most recent, full database backup image. The default operation of the PRUNE HISTORY command prevents required entries from being deleted. It is recommended that the FORCE option of the PRUNE HISTORY command not be used.

It is a good idea to keep a complete record of backup operations, and to use this utility as a guide.

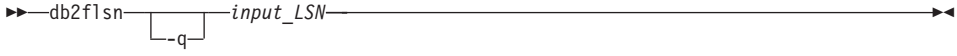
db2flsn - Find Log Sequence Number

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
- -500 Logical error.

Other valid return codes are:

- 0 Successful execution
- 99 Warning: the result is based on the last known log file size.

input_LSN
A 12-byte string that represents the internal (6-byte) hexadecimal value with leading zeros.

Examples

```
db2flsn 000000BF0030
   Given LSN is contained in log file S0000002.LOG

db2flsn -q 000000BF0030
   S0000002.LOG

db2flsn 000000BE0030
   Warning: the result is based on the last known log file size.
   The last known log file size is 23 4K pages starting from log extent 2.
```

```
   Given LSN is contained in log file S0000001.LOG

db2flsn -q 000000BE0030
   S0000001.LOG
```

Usage Notes

The log header control file `sqlogctl.lfh` must reside in the current directory. Since this file is located in the database directory, the tool can be run from the database directory, or the control file can be copied to the directory from which the tool will be run.

The tool uses the *logfilsiz* database configuration parameter. DB2 records the three most recent values for this parameter, and the first log file that is created with each *logfilsiz* value; this enables the tool to work correctly when *logfilsiz* changes. If the specified LSN predates the earliest recorded value of *logfilsiz*, the tool uses this value, and returns a warning. The tool can be used with database managers prior to UDB Version 5.2; in this case, the warning is returned even with a correct result (obtained if the value of *logfilsiz* remains unchanged).

This tool can only be used with recoverable databases. A database is recoverable if it is configured with *logretain* set to `RECOVERY` or *userexit* set to `ON`.

db2inidb - Initialize a Mirrored Database

In a split mirror environment, this command is used to initialize a mirrored database for different purposes.

Authorization

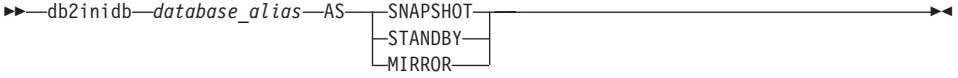
One of the following:

- *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. This database is read-only.

STANDBY
Specifies that the database will be placed in rollforward 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 that can be used to restore the primary database.

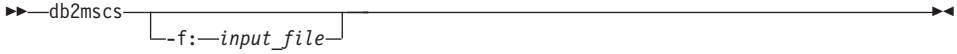
db2mscs - Set up Windows NT Failover Utility

Creates the infrastructure for DB2 failover support on Windows NT/2000 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.

ARCHIVE LOG

Closes and truncates the active log file for a recoverable database. If user exit is enabled, issues an archive request.

Scope

In an MPP environment, this command closes and truncates the active logs on all nodes; however, a subset of nodes can be specified.

Authorization

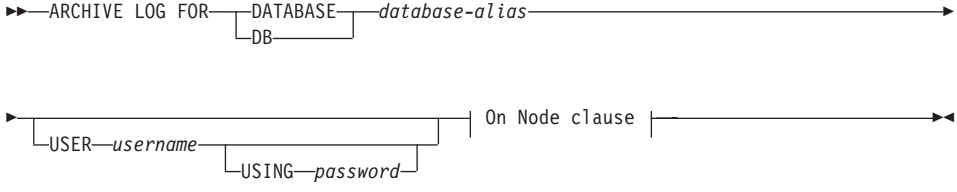
One of the following:

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

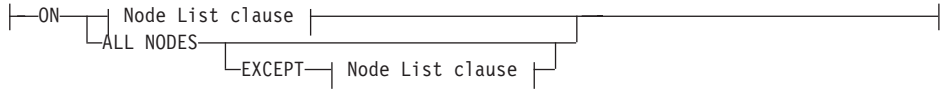
Required Connection

This command automatically establishes a connection to the specified database. If a connection already exists, an error is returned.

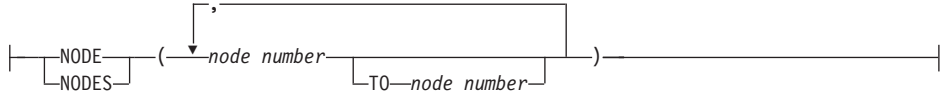
Command Syntax



On Node clause:



Node 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 NODES
Specifies that the command should be issued on all nodes in the `db2nodes.cfg` file. This is the default if a node clause is not specified.

EXCEPT
Specifies that the command should be issued on all nodes in the `db2nodes.cfg` file, except those specified in the node list.

ON NODE/ON NODES
Specifies that the logs should be archived for the specified database on a set of nodes.

node number
Specifies a node number in the node list.

TO node number
Used when specifying a range of nodes for which the logs should be archived. All nodes from the first node number specified up to and including the second node number specified are included in the node 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.

If other applications have transactions in progress when this command is invoked, a slight performance decrement will be noticed when the log buffer is flushed to disk; other transactions attempting to write log records to the buffer must wait until the flush has completed.

This command causes the database to lose a portion of its LSN space, thereby hastening the exhaustion of valid LSNs.

INITIALIZE TAPE

DB2 for Windows NT/2000 supports backup and restore operations to streaming tape devices. Use this command for tape initialization.

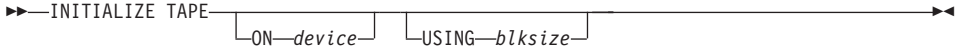
Authorization

None

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.

Note: The buffer size specified in "BACKUP DATABASE Command" on page 84, and in "RESTORE DATABASE Command" on page 110 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.

See Also

"REWIND TAPE" on page 323

"SET TAPE POSITION" on page 324.

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, or rename table space, run statistics, reorganize table, drop table, and load.

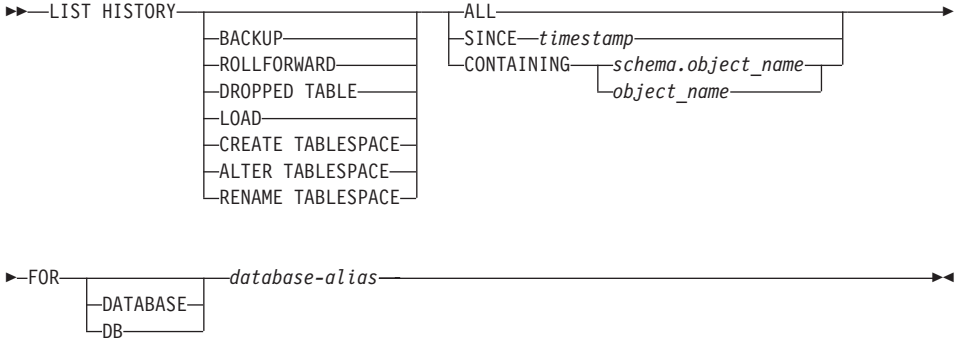
Authorization

None

Required Connection

Instance. An explicit attachment is not required. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the command.

Command Syntax



Command Parameters

HISTORY
Lists all events that are currently logged in the history file.

BACKUP
Lists backup and restore operations.

ROLLFORWARD
Lists rollforward operations.

DROPPED TABLE
Lists dropped table records.

LOAD
Lists load operations.

CREATE TABLESPACE
Lists table space create and drop operations.

RENAME TABLESPACE
Lists table space renaming operations.

ALTER TABLESPACE
Lists alter table space operations.

ALL Lists all entries of the specified type in the history file.

SINCE timestamp
A complete time stamp (format *yyyymmddhhmnss*), or an initial prefix (minimum *yyyy*) can be specified. All entries with time stamps equal to or greater than the time stamp provided are listed.

CONTAINING schema.object_name
This qualified name uniquely identifies a table.

CONTAINING object_name
This unqualified name uniquely identifies a table space.

FOR DATABASE database-alias
Used to identify the database whose recovery history file is to be listed.

Examples

```
db2 list history since 19980201 for sample
db2 list history backup containing userspace1 for sample
db2 list history dropped table all for db sample
```

Usage Notes

The report generated by this command contains the following symbols:

```
Operation

   A - Create table space
   B - Backup
   C - Load copy
   D - Dropped table
   F - Roll forward
   G - Reorganize table
   L - Load
   N - Rename table space
   O - Drop table space
   Q - Quiesce
   R - Restore
   S - Run statistics
   T - Alter table space
   U - Unload

Type

Backup types:

   F - Offline
   N - Online
```

```
   I - Incremental offline
   O - Incremental online
   D - Delta offline
   E - Delta online

Rollforward types:

   E - End of logs
   P - Point in time

Load types:

   I - Insert
   R - Replace

Alter tablespace types:

   C - Add containers
   R - Rebalance

Quiesce types:

   S - Quiesce share
   U - Quiesce update
   X - Quiesce exclusive
   Z - Quiesce reset
```

If a rollforward operation has been carried out to the end of all the logs, the backup ID represents the end of time; that is, the backup ID value is 99991231235959.

PRUNE HISTORY/LOGFILE

Used to delete entries from the recovery history file, or to delete log files from the active log file path. Deleting entries from the recovery history file may be necessary if the file becomes excessively large and the retention period is high. Deleting log files from the active log file path may be necessary if logs are being archived manually (rather than through a user exit program).

Authorization

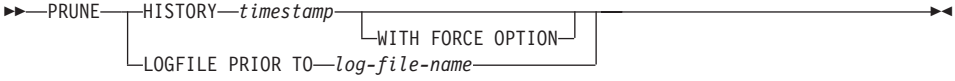
One of the following:

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

Required Connection

Database

Command Syntax



Command Parameters

HISTORY timestamp

Identifies a range of entries in the recovery history file that will be deleted. A complete time stamp (in the form *yyyymmddhhmmss*), or an initial prefix (minimum *yyyy*) can be specified. All entries with time stamps equal to or less than the time stamp provided are deleted from the recovery history file.

WITH FORCE OPTION

Specifies that the entries will be pruned according to the time stamp specified, even if some entries from the most recent restore set are deleted from the file. A restore set is the most recent full database backup including any restores of that backup image. If this parameter is not specified, all entries from the backup image forward will be maintained in the history.

LOGFILE PRIOR TO log-file-name

Specifies a string for a log file name, for example `S0000100.LOG`. All log files prior to (but not including) the specified log file will be deleted. The LOGRETAIN database configuration parameter must be set to `RECOVERY` or `CAPTURE`.

Examples

To remove the entries for all restores, loads, table space backups, and full database backups taken before and including December 1, 1994 from the recovery history file, enter:

```
db2 prune history 199412
```

Note: 199412 is interpreted as 19941201000000.

Usage Notes

Pruning backup entries from the history file causes related file backups on DB2 Data Links Manager servers to be deleted.

REWIND TAPE

DB2 for Windows NT/2000 supports backup and restore operations to streaming tape devices. Use this command for tape rewinding.

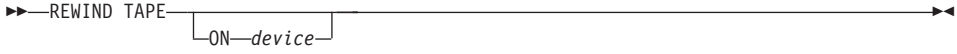
Authorization

None

Required Connection

None

Command Syntax



Command Parameters

ON device
Specifies a valid tape device name. The default value is `\\.\TAPE0`.

See Also

"INITIALIZE TAPE" on page 317

"SET TAPE POSITION" on page 324.

SET TAPE POSITION

DB2 for Windows NT/2000 supports backup and restore operations to streaming tape devices. Use this command for tape positioning.

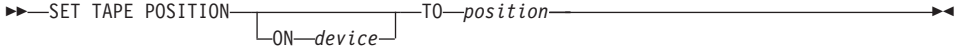
Authorization

None

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 NT/2000 writes a tape mark after every backup image. A value of 1 specifies the first position, 2 specifies the second position, and so on. If the tape is positioned at tape mark 1, for example, archive 2 is positioned to be restored.

See Also

"INITIALIZE TAPE" on page 317

"REWIND TAPE" on page 323.

UPDATE HISTORY FILE

Updates the location, device type, or comment in a history file entry.

Authorization

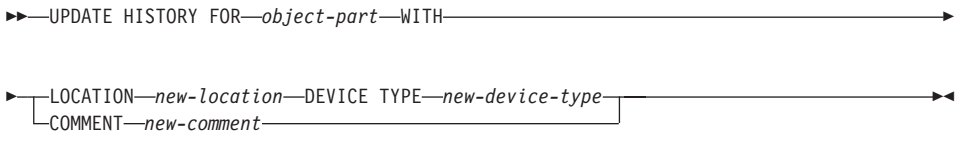
One of the following:

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

Required Connection

Database

Command Syntax



Command Parameters

FOR object-part
Specifies the identifier for the backup or copy image. It is a time stamp with an optional sequence number from `001` to `999`.

LOCATION new-location
Specifies the new physical location of a backup image. The interpretation of this parameter depends on the device type.

DEVICE TYPE new-device-type
Specifies a new device type for storing the backup image. Valid device types are:

D Disk

K Diskette

T Tape

A TSM

U User exit

O Other

COMMENT new-comment
Specifies a new comment to describe the entry.

Examples

To update the history file entry for a full database backup taken on April 13, 1997 at 10:00 a.m., enter:

```
db2 update history for 19970413100000001 with
   location /backup/dbbackup.1 device type d
```

Usage Notes

The history file is used by database administrators for record keeping. It is used internally by DB2 for the automatic recovery of incremental backups.

See Also

"PRUNE HISTORY/LOGFILE" on page 321.

Appendix D. Additional APIs and Associated Data Structures

db2ArchiveLog - Archive Active Log API

Closes and truncates the active log file for a recoverable database. If user exit is enabled, issues an archive request.

Scope

In an MPP environment, this API closes and truncates the active logs on all nodes.

Authorization

One of the following:

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

Required Connection

This API automatically establishes a connection to the specified database. If a connection already exists, an error is returned.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Archive Active Log */
/* ... */
SQL_API_RC SQL_API_FN
   db2ArchiveLog (
      db2Uint32 version,
      void * pDB2ArchiveLogStruct,
      struct sqlca * pSqlca);

typedef struct
{
   char * piDatabaseAlias;
   char * piUserName;
   char * piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE * piNodeList;
   db2Uint32 iOptions;
} db2ArchiveLogStruct;
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Archive Active Log */
/* ... */
SQL_API_RC SQL_API_FN
   db2gArchiveLog (
      db2Uint32 version,
      void * pDB2gArchiveLogStruct,
      struct sqlca * pSqlca);

typedef struct
{
   db2Uint32 iAliasLen;
   db2Uint32 iUserNameLen;
   db2Uint32 iPasswordLen;
   char * piDatabaseAlias;
   char * piUserName;
   char * piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE * piNodeList;
   db2Uint32 iOptions;
} db2gArchiveLogStruct;
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the variable passed in as the second parameter, *pDB2ArchiveLogStruct*.

pDB2ArchiveLogStruct
Input. A pointer to the *db2ArchiveLogStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

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

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

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

piDatabaseAlias
Input. A string containing the database alias (as cataloged in the system database directory) of the database for which the active log is to be archived.

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

piPassword
Input. A string containing the password to be used when attempting a connection.

iAllNodeFlag
Input. MPP only. Flag indicating whether the operation should apply to all nodes listed in the `db2nodes.cfg` file. Valid values are:

DB2ARCHIVELOG_ALL_NODES
Apply to all nodes (piNodeList should be NULL). This is the default value.

DB2ARCHIVELOG_NODE_LIST
Apply to all nodes specified in a node list that is passed in *piNodeList*.

DB2ARCHIVELOG_ALL_EXCEPT
Apply to all nodes *except* those specified in a node list that is passed in *piNodeList*.

iNumNodes
Input. MPP only. Specifies the number of nodes in the *piNodeList* array.

piNodeList
Input. MPP only. A pointer to an array of node numbers against which to apply the archive log operation.

iOptions
Input. Reserved for future use.

Usage Notes

This API can be used to collect a complete set of log files up to a known point. The log files can then be used to update a standby database.

If other applications have transactions in progress when this API is called, a slight performance decrement will be noticed when the log buffer is flushed to disk; other transactions attempting to write log records to the buffer must wait until the flush has completed.

This API causes the database to lose a portion of its LSN space, thereby hastening the exhaustion of valid LSNs.

db2HistoryCloseScan - Close Recovery History File Scan API

Ends a recovery history file scan and frees DB2 resources required for the scan. This API must be preceded by a successful call to "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

Authorization

None

Required Connection

Instance. It is not necessary to call **sqleatin** before calling this API.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Close Recovery History File Scan */
/* ... */
SQL_API_RC  SQL_API_FN
  db2HistoryCloseScan (
    db2Uint32 version,
    void * piHandle,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Close Recovery History File Scan */
/* ... */
SQL_API_RC  SQL_API_FN
  db2GenHistoryCloseScan (
    db2Uint32 version,
    void * piHandle,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the second parameter, *piHandle*.

piHandle
Input. Specifies a pointer to the handle for scan access that was returned by "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

REXX API Syntax

```
CLOSE RECOVERY HISTORY FILE :scanid
```

REXX API Parameters

scanid Host variable containing the scan identifier returned from OPEN RECOVERY HISTORY FILE SCAN.

Usage Notes

For a detailed description of the use of the recovery history file APIs, see "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

See Also

"db2HistoryGetEntry - Get Next Recovery History File Entry API" on page 333

"db2HistoryOpenScan - Open Recovery History File Scan API" on page 337

"db2Prune API" on page 345

"db2HistoryUpdate - Update Recovery History File API" on page 342.

db2HistoryGetEntry - Get Next Recovery History File Entry API

Gets the next entry from the recovery history file. This API must be preceded by a successful call to "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

Authorization

None

Required Connection

Instance. It is not necessary to call **sqleatin** before calling this API.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Get Next Recovery History File Entry */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryGetEntry (
    db2Uint32 version,
    void * pDB2HistoryGetEntryStruct,
    struct sqlca * pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData * pioHistData
} db2HistoryGetEntryStruct;
/* ... */
```

db2HistoryGetEntry - Get Next Recovery History File Entry API

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Get Next Recovery History File Entry */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryGetEntry (
    db2Uint32 version,
    void * pDB2GenHistoryGetEntryStruct,
    struct sqlca * pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData * pioHistData
} db2GenHistoryGetEntryStruct;
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryGetEntryStruct*.

pDB2HistoryGetEntryStruct
Input. A pointer to the *db2HistoryGetEntryStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

iHandle
Input. Contains the handle for scan access that was returned by "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

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

DB2HISTORY_GET_ENTRY
Get the next entry, but without any command data.

DB2HISTORY_GET_DDL
Get only the command data from the previous fetch.

DB2HISTORY_GET_ALL
Get the next entry, including all data.

pioHistData
Input. A pointer to the *db2HistData* structure. For more information about this structure, see "Data Structure: db2HistData" on page 352.

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 recovery history file entry information is returned. In the following, XXX represents the host variable name:

XXX.0	Number of first level elements in the variable (always 15)
XXX.1	Number of table space elements
XXX.2	Number of used table space elements
XXX.3	OPERATION (type of operation performed)
XXX.4	OBJECT (granularity of the operation)
XXX.5	OBJECT_PART (time stamp and sequence number)
XXX.6	OPTYPE (qualifier of the operation)
XXX.7	DEVICE_TYPE (type of device used)
XXX.8	FIRST_LOG (earliest log ID)
XXX.9	LAST_LOG (current log ID)
XXX.10	BACKUP_ID (identifier for the backup)
XXX.11	SCHEMA (qualifier for the table name)
XXX.12	TABLE_NAME (name of the loaded table)
XXX.13.0	NUM_OF_TABLESPACES (number of table spaces involved in backup or restore)
XXX.13.1	Name of the first table space backed up/restored
XXX.13.2	Name of the second table space backed up/restored
XXX.13.3	and so on
XXX.14	LOCATION (where backup or copy is stored)
XXX.15	COMMENT (text to describe the entry).

Usage Notes

The records that are returned will have been selected using the values specified on the call to "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

For a detailed description of the use of the recovery history file APIs, see "db2HistoryOpenScan - Open Recovery History File Scan API" on page 337.

See Also

"db2HistoryCloseScan - Close Recovery History File Scan API" on page 331

"db2HistoryOpenScan - Open Recovery History File Scan API" on page 337

"db2Prune API" on page 345

"db2HistoryUpdate - Update Recovery History File API" on page 342.

db2HistoryOpenScan - Open Recovery History File Scan API

Starts a recovery history file scan.

Authorization

None

Required Connection

Instance. It is not necessary to call **sqleatin** before calling this API. If the database is cataloged as remote, an instance attachment to the remote node is established.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Open Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryOpenScan (
    db2Uint32 version,
    void * pDB2HistoryOpenStruct,
    struct sqlca * pSqlca);

typedef struct
{
  char * piDatabaseAlias,
  char * piTimestamp,
  char * piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2HistoryOpenStruct;
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Open Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryOpenScan (
    db2Uint32 version,
    void * pDB2GenHistoryOpenStruct,
    struct sqlca * pSqlca);

typedef struct
{
  char * piDatabaseAlias,
  char * piTimestamp,
  char * piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2GenHistoryOpenStruct;
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryOpenStruct*.

pDB2HistoryOpenStruct
Input. A pointer to the *db2HistoryOpenStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

piDatabaseAlias
Input. A pointer to a string containing the database alias.

piTimestamp
Input. A pointer to a string specifying the time stamp to be used for selecting records. Records whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL, or pointing to zero, prevents the filtering of entries using a time stamp.

piObjectName
Input. A pointer to a string specifying the object name to be used for selecting records. The object may be a table or a table space. If it is a table, the fully qualified table name must be provided. Setting this parameter to NULL, or pointing to zero, prevents the filtering of entries using the object name.

oNumRows
Output. Upon return from the API, this parameter contains the number of matching recovery history file entries.

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

DB2HISTORY_LIST_HISTORY
Lists all events that are currently logged in the history file.

DB2HISTORY_LIST_BACKUP
Lists backup and restore operations.

DB2HISTORY_LIST_ROLLFORWARD
Lists rollforward operations.

DB2HISTORY_LIST_DROPPED_TABLE
Lists dropped table records. The DDL field associated with an entry is not returned. To retrieve the DDL information for an entry, "db2HistoryGetEntry - Get Next Recovery History File Entry API" on page 333 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 - Get Next Recovery History File Entry API" on page 333 must be called with a caller action of `DB2HISTORY_GET_DDL` immediately after the entry is fetched.

DB2HISTORY_LIST_RUNSTATS
Lists RUNSTATS operations. This value is not currently supported.

DB2HISTORY_LIST_REORG
Lists REORGANIZE TABLE operations. This value is not currently supported.

oHandle
Output. Upon return from the API, this parameter contains the handle for scan access. It is subsequently used in "db2HistoryGetEntry - Get

Next Recovery History File Entry API" on page 333, and "db2HistoryCloseScan - Close Recovery History File Scan API" on page 331.

REXX API Syntax

```
OPEN [BACKUP] RECOVERY HISTORY FILE FOR database_alias
[OBJECT objname] [TIMESTAMP :timestamp]
USING :value
```

REXX API Parameters

database_alias
The alias of the database whose history file is to be listed.

objname
Specifies the object name to be used for selecting records. The object may be a table or a table space. If it is a table, the fully qualified table name must be provided. Setting this parameter to NULL prevents the filtering of entries using *objname*.

timestamp
Specifies the time stamp to be used for selecting records. Records whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL prevents the filtering of entries using *timestamp*.

value A compound REXX host variable to which recovery 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 recovery history file entries.

Usage Notes

The combination of time stamp, object name and caller action can be used to filter records. Only records that pass all specified filters are returned.

The filtering effect of the object name depends on the value specified:

- Specifying a table will return records for load operations, because this is the only information for tables in the history file.
- Specifying a table space will return records for backup, restore, and load operations for the table space.

Note: To return records for tables, they must be specified as *schema.tablename*. Specifying *tablename* will only return records for table spaces.

A maximum of eight history file scans per process is permitted.

To list every entry in the history file, a typical application will perform the following steps:

1. Call **db2HistoryOpenScan**, which will return *oNumRows*.
2. Allocate an *db2HistData* structure with space for *n oTablespace* fields, where *n* is an arbitrary number.
3. Set the *iDB2NumTablespace* field of the *db2HistData* structure to *n*.
4. In a loop, perform the following:
   - Call **db2HistoryGetEntry** to fetch from the history file.
   - If **db2HistoryGetEntry** returns an SQLCODE of `SQL_RC_OK`, use the *sqld* field of the *db2HistData* structure to determine the number of table space entries returned.
   - If **db2HistoryGetEntry** returns an SQLCODE of `SQLUH_SQLUHINFO_VARS_WARNING`, not enough space has been allocated for all of the table spaces that DB2 is trying to return; free and reallocate the *db2HistData* structure with enough space for *oDB2UsedTablespace* table space entries, and set *iDB2NumTablespace* to *oDB2UsedTablespace*.
   - If **db2HistoryGetEntry** returns an SQLCODE of `SQLE_RC_NOMORE`, all recovery history file entries have been retrieved.
   - Any other SQLCODE indicates a problem.
5. When all of the information has been fetched, call "db2HistoryCloseScan - Close Recovery History File Scan API" on page 331 to free the resources allocated by the call to **db2HistoryOpenScan**.

The macro SQLUHINFOSIZE(*n*), defined in `sqlutil`, is provided to help determine how much memory is required for an *db2HistData* structure with space for *n oTablespace* fields.

See Also

"db2HistoryCloseScan - Close Recovery History File Scan API" on page 331

"db2HistoryGetEntry - Get Next Recovery History File Entry API" on page 333

"db2Prune API" on page 345

"db2HistoryUpdate - Update Recovery History File API" on page 342.

db2HistoryUpdate - Update Recovery History File API

Updates the location, device type, or comment in a history file entry.

Authorization

One of the following:

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

Required Connection

Database. To update entries in the history file for a database other than the default database, a connection to the database must be established before calling this API.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Update Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryUpdate (
    db2Uint32 version,
    void * pDB2HistoryUpdateStruct,
    struct sqlca * pSqlca);

typedef struct
{
  char * piNewLocation,
  char * piNewDeviceType,
  char * piNewComment,
  db2Uint32 iEID
} db2HistoryUpdateStruct;
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Update Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryUpdate (
    db2Uint32 version,
    void * pDB2GenHistoryUpdateStruct,
    struct sqlca * pSqlca);

typedef struct
{
  char * piNewLocation,
  char * piNewDeviceType,
  char * piNewComment,
  db2Uint32 iEID
} db2GenHistoryUpdateStruct;
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryUpdateStruct*.

pDB2HistoryUpdateStruct
Input. A pointer to the *db2HistoryUpdateStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

piNewLocation
Input. A pointer to a string specifying a new location for the backup, restore, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

piNewDeviceType
Input. A pointer to a string specifying a new device type for storing the backup, restore, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

piNewComment
Input. A pointer to a string specifying a new comment to describe the entry. Setting this parameter to NULL, or pointing to zero, leaves the comment unchanged.

iEID Input. A unique identifier that can be used to update a specific entry in the history file.

REXX API Syntax

```
UPDATE RECOVERY HISTORY USING :value
```

REXX API Parameters

value A compound REXX host variable containing information pertaining to the new location of a recovery history file entry. In the following, XXX represents the host variable name:

XXX.0 Number of elements in the variable (must be between 1 and 4)

XXX.1 OBJECT_PART (time stamp with a sequence number from 001 to 999)

XXX.2 New location for the backup or copy image (this parameter is optional)

XXX.3 New device used to store the backup or copy image (this parameter is optional)

XXX.4 New comment (this parameter is optional).

Usage Notes

This is an update function, and all information prior to the change is replaced and cannot be recreated. These changes are not logged.

The history file is used for recording purposes only. It is not used directly by the restore or the rollforward functions. During a restore operation, the location of the backup image can be specified, and the history file is useful for tracking this location. The information can subsequently be provided to the backup utility (see "Backup Database API" on page 88). Similarly, if the location of a load copy image is moved, the rollforward utility must be provided with the new location and type of storage media. For additional information, see "Rollforward Database API" on page 148.

See Also

"db2HistoryCloseScan - Close Recovery History File Scan API" on page 331

"db2HistoryGetEntry - Get Next Recovery History File Entry API" on page 333

"db2HistoryOpenScan - Open Recovery History File Scan API" on page 337

"db2Prune API" on page 345.

db2Prune API

Deletes entries from the recovery 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 recovery history file for any database other than the default database, a connection to the database must be established before calling this API.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Prune Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2Prune (
    db2Uint32 version,
    void * pDB2PruneStruct,
    struct sqlca * pSqlca);

typedef struct
{
  char * piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2PruneStruct;
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Prune Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenPrune (
    db2Uint32 version,
    void * pDB2GenPruneStruct,
    struct sqlca * pSqlca);

typedef struct
{
  db2Uint32 iStringLen;
  char * piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2GenPruneStruct;
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2PruneStruct*.

pDB2PruneStruct
Input. A pointer to the *db2PruneStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

iStringLen
Input. Specifies the length in bytes of *piString*.

piString
Input. A pointer to a string specifying a time stamp or a log sequence number (LSN). The time stamp or part of a time stamp (minimum *yyyy*, or year) is used to select records for deletion. All entries equal to or less than the time stamp will be deleted. A valid time stamp must be provided; there is no default behavior for a NULL parameter.

This parameter can also be used to pass an LSN, so that inactive logs can be pruned.

iEID Input. Specifies a unique identifier that can be used to prune a single entry from the history file.

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

DB2PRUNE_ACTION_HISTORY
Remove history file entries.

DB2PRUNE_ACTION_LOG
Remove log files from the active log path.

iOptions
Input. Valid values (defined in `db2ApiDf`) are:

DB2PRUNE_OPTION_FORCE
Force the removal of the last backup.

DB2PRUNE_OPTION_LSNSTRING
Specify that the value of *piString* is an LSN, used when a caller action of `DB2PRUNE_ACTION_LOG` is specified.

REXX API Syntax

```
PRUNE RECOVERY HISTORY BEFORE :timestamp [WITH FORCE OPTION]
```

REXX API Parameters

timestamp
A host variable containing a time stamp. All entries with time stamps equal to or less than the time stamp provided are deleted from the recovery history file.

WITH FORCE OPTION
If specified, the recovery history file will be pruned according to the time stamp specified, even if some entries from the most recent restore set are deleted from the file. If not specified, the most recent restore set will be kept, even if the time stamp is less than or equal to the time stamp specified as input.

Usage Notes

Pruning the history file does not delete the actual backup or load files. The user must manually delete these files to free up the space they consume on storage media.

CAUTION:
If the latest full database backup is deleted from the media (in addition to being pruned from the history file), the user must ensure that all table spaces, including the catalog table space and the user table spaces, are backed up. Failure to do so may result in a database that cannot be recovered, or the loss of some portion of the user data in the database.

See Also

"db2HistoryCloseScan - Close Recovery History File Scan API" on page 331

"db2HistoryGetEntry - Get Next Recovery History File Entry API" on page 333

"db2HistoryOpenScan - Open Recovery History File Scan API" on page 337

"db2HistoryUpdate - Update Recovery History File API" on page 342.

sqlurlog - Asynchronous Read Log API

Provides the caller with the ability to extract certain log records from the database logs, and to query the Log Manager for current log state information. This API can only be used with recoverable databases. A database is recoverable if it is configured with *logretain* set to RECOVERY or *userexit* set to ON.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Asynchronous Read Log */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurlog (
    sqluint32 CallerAction,
    SQLU_LSN * pStartLsn,
    SQLU_LSN * pEndLsn,
    char * pLogBuffer,
    sqluint32 LogBufferSize,
    SQLU_RLOG_INFO * pReadLogInfo,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

CallerAction
Input. Specifies the action to be performed.

SQLU_RLOG_READ
Read the database log from the starting log sequence to the ending log sequence number and return all propagatable log records within this range.

SQLU_RLOG_READ_SINGLE
Read a single log record (propagatable or not) identified by the starting log sequence number.

SQLU_RLOG_QUERY
Query the database log. Results of the query will be sent back

via the SQLU_RLOG_INFO structure (see "Data Structure: SQLU-RLOG-INFO" on page 358).

pStartLsn
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.

pEndLsn
Input. The ending log sequence number specifies the ending relative byte address for the reading of the log. This value must be greater than *startLsn*, and does not need to be the end of an actual log record.

pLogBuffer
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.

LogBufferSize
Output. Specifies the size, in bytes, of the log buffer.

pReadLogInfo
Output. A structure detailing information regarding the call and the database log. For more information about this structure, see "Data Structure: SQLU-RLOG-INFO" on page 358.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* or the *SQL Reference*.

Sample Programs

C \sqllib\samples\c\asynrlog.sqc

Usage Notes

If the requested action is to read the log, the caller will provide a log sequence number range and a buffer to hold the log records. This API reads the log sequentially, bounded by the requested LSN range, and returns log records associated with tables having the DATA CAPTURE option CHANGES, and an SQLU_RLOG_INFO structure with the current active log information. If the requested action is query, the API returns an SQLU_RLOG_INFO 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 (SQLU-RLOG-INFO) will contain a valid starting LSN (in the initialLSN

member), to be used on a read call. The end of the current active log will be in the curActiveLSN member of the read log information structure. The value used as the ending LSN on a read can be one of the following:

- The value of the curActiveLSN
- A value greater than initialLSN
- FFFF FFFF FFFF, which is interpreted by the asynchronous log reader as the end of the current log.

For more information about the read log information structure, see "Data Structure: SQLU-RLOG-INFO" on page 358.

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 **sqlurlog** can be found in the *Administrative API Reference*.

To read the next sequential log record after the initial read, add 1 to the last read LSN returned in SQLU-RLOG-INFO. 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.

Data Structure: db2HistData

This structure is used to return information after a call to "db2HistoryGetEntry - Get Next Recovery History File Entry API" on page 333.

Table 17. Fields in the db2HistData Structure

Field Name	Data Type	Description
ioHistDataID	char(8)	An 8-byte structure identifier and "eye-catcher" for storage dumps. The only valid value is "SQLUHINF". No symbolic definition for this string exists.
oObjectPart	db2Char	The first 14 characters are a time stamp with format *yyyymmddhhnnss*, indicating when the operation was begun. The next 3 characters are a sequence number. Each backup operation can result in multiple entries in this file when the backup image is saved in multiple files or on multiple tapes. The sequence number allows multiple locations to be specified. Restore and load operations have only a single entry in this file, which corresponds to sequence number '001' of the corresponding backup. The time stamp, combined with the sequence number, must be unique.
oEndTime	db2Char	A time stamp with format *yyyymmddhhnnss*, indicating when the operation was completed.
oFirstLog	db2Char	The earliest log file ID (ranging from S0000000 to S9999999): • Required to apply rollforward recovery for an online backup • Required to apply rollforward recovery for an offline backup • Applied after restoring a full database or table space level backup that was current when the load started.
oLastLog	db2Char	The latest log file ID (ranging from S0000000 to S9999999): • Required to apply rollforward recovery for an online backup • Required to apply rollforward recovery to the current point in time for an offline backup • Applied after restoring a full database or table space level backup that was current when the load operation finished (will be the same as *oFirstLog* if roll forward recovery is not applied).

Table 17. Fields in the db2HistData Structure (continued)

Field Name	Data Type	Description
oID	db2Char	Unique backup or table identifier.
oTableQualifier	db2Char	Table qualifier.
oTableName	db2Char	Table name.
oLocation	db2Char	For backups and load copies, this field indicates where the data has been saved. For operations that require multiple entries in the file, the sequence number defined by *oObjectPart* identifies which part of the backup is found in the specified location. For restore and load operations, the location always identifies where the first part of the data restored or loaded (corresponding to sequence '001' for multi-part backups) has been saved. The data in *oLocation* is interpreted differently, depending on *oDeviceType*: • For disk or diskette (`D` or `K`), a fully qualified file name • For tape (`T`), a volume label • For TSM (`A`), the server name • For user exit or other (`U` or `O`), free form text.
oComment	db2Char	Free form text comment.
oCommandText	db2Char	Command text, or DDL.
oLastLSN	SQLU_LSN	Last log sequence number.
oEID	Structure	Unique entry identifier.
poEventSQLCA	Structure	Result *sqlca* of the recorded event. For information about the *sqlca* structure, see the *Administrative API Reference* and the *SQL Reference*.
poTablespace	db2Char	A list of table space names.
ioNumTablespaces	db2Uint32	Number of entries in the *poTablespace* list. Each table space backup contains one or more table spaces. Each table space restore operation replaces one or more table spaces. If this field is not zero (indicating a table space level backup or restore), the next lines in this file contain the name of the table space backed up or restored, represented by an 18-character string. One table space name appears on each line.
oOperation	char	See Table 18 on page 354.
oObject	char	Granularity of the operation: `D` for full database, `P` for table space, and `T` for table.
oOptype	char	See Table 19 on page 355.

Table 17. Fields in the db2HistData Structure (continued)

Field Name	Data Type	Description
oStatus	char	Entry status: `D` for deleted (future use), `E` for expired, `I` for inactive, `N` for not yet committed, and `Y` for committed or active.
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, `S` for server, `T` for tape, and `U` for user exit.

Table 18. Valid oOperation Values in the db2HistData Structure

Value	Description	C Definition	COBOL/FORTRAN Definition
A	add table space	DB2HISTORY_OP_ADD_ TABLESPACE	DB2HIST_OP_ADD_ TABLESPACE
B	backup	DB2HISTORY_OP_BACKUP	DB2HIST_OP_BACKUP
C	load copy	DB2HISTORY_OP_LOAD_COPY	DB2HIST_OP_LOAD_COPY
D	dropped table	DB2HISTORY_OP_DROPPED_ TABLE	DB2HIST_OP_DROPPED_TABLE
F	rollforward	DB2HISTORY_OP_ROLLFWD	DB2HIST_OP_ROLLFWD
G	reorganize table	DB2HISTORY_OP_REORG	DB2HIST_OP_REORG
L	load	DB2HISTORY_OP_LOAD	DB2HIST_OP_LOAD
N	rename table space	DB2HISTORY_OP_REN_ TABLESPACE	DB2HIST_OP_REN_ TABLESPACE
O	drop table space	DB2HISTORY_OP_DROP_ TABLESPACE	DB2HIST_OP_DROP_ TABLESPACE
Q	quiesce	DB2HISTORY_OP_QUIESCE	DB2HIST_OP_QUIESCE
R	restore	DB2HISTORY_OP_RESTORE	DB2HIST_OP_RESTORE
S	run statistics	DB2HISTORY_OP_RUNSTATS	DB2HIST_OP_RUNSTATS
T	alter table space	DB2HISTORY_OP_ALT_ TABLESPACE	DB2HIST_OP_ALT_TBS
U	unload	DB2HISTORY_OP_UNLOAD	DB2HIST_OP_UNLOAD

Table 19. Valid oOptype Values in the db2HistData Structure

oOperation	oOptype	Description	C/COBOL/FORTRAN Definition
B	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
	I	incremental offline	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	incremental online	DB2HISTORY_OPTYPE_INCR_ ONLINE
	D	delta offline	DB2HISTORY_OPTYPE_DELTA_ OFFLINE
	E	delta online	DB2HISTORY_OPTYPE_DELTA_ ONLINE
F	E	end of logs	DB2HISTORY_OPTYPE_EOL
	P	point in time	DB2HISTORY_OPTYPE_PIT
L	I	insert	DB2HISTORY_OPTYPE_INSERT
	R	replace	DB2HISTORY_OPTYPE_REPLACE
Q	S	quiesce share	DB2HISTORY_OPTYPE_SHARE
	U	quiesce update	DB2HISTORY_OPTYPE_UPDATE
	X	quiesce exclusive	DB2HISTORY_OPTYPE_EXCL
	Z	quiesce reset	DB2HISTORY_OPTYPE_RESET
R	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
	I	incremental offline	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	incremental online	DB2HISTORY_OPTYPE_INCR_ ONLINE
T	C	add containers	DB2HISTORY_OPTYPE_ADD_CONT
	R	rebalance	DB2HISTORY_OPTYPE_REB

Table 20. Fields in the db2Char Structure

Field Name	Data Type	Description
pioData	char	A pointer to a character data buffer. If NULL, no data will be returned.
iLength	db2Uint32	Input. The size of the *pioData* buffer.
oLength	db2Uint32	Output. The number of valid characters of data in the *pioData* buffer.

Table 21. Fields in the db2HistoryEID Structure

Field Name	Data Type	Description
ioNode	SQL_PDB_NODE_TYPE	Node number.
ioHID	db2Uint32	Local history file entry ID.

Data Structure: db2HistData

Language Syntax

C Structure

```
/* File: db2ApiDf.h */
/* ... */
typedef SQL_STRUCTURE db2HistoryData
{
  char ioHistDataID[8];
  db2Char oObjectPart;
  db2Char oEndTime;
  db2Char oFirstLog;
  db2Char oLastLog;
  db2Char oID;
  db2Char oTableQualifier;
  db2Char oTableName;
  db2Char oLocation;
  db2Char oComment;
  db2Char oCommandText;
  SQLU_LSN oLastLSN;
  db2HistoryEID oEID;
  struct sqlca * poEventSQLCA;
  db2Char * poTablespace;
  db2Uint32 ioNumTablespaces;
  char oOperation;
  char oObject;
  char oOptype;
  char oStatus;
  char oDeviceType
} db2HistoryData;

typedef SQL_STRUCTURE db2Char
{
  char * pioData;
  db2Uint32 ioLength
} db2Char;

typedef SQL_STRUCTURE db2HistoryEID
{
  SQL_PDB_NODE_TYPE ioNode;
  db2Uint32 ioHID
} db2HistoryEID;
/* ... */
```

Data Structure: SQLU-LSN

This union, used by "sqlurlog - Asynchronous Read Log API" on page 349, contains the definition of the log sequence number. A log sequence number (LSN) represents a relative byte address within the database log. All log records are identified by this number. It represents the log record's byte offset from the beginning of the database log.

Table 22. Fields in the SQLU-LSN Union

Field Name	Data Type	Description
lsnChar	Array of UNSIGNED CHAR	Specifies the 6-member character array log sequence number.
lsnWord	Array of UNSIGNED SHORT	Specifies the 3-member short array log sequence number.

Language Syntax

C Structure

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

Data Structure: SQLU-RLOG-INFO

This structure contains information about the status of calls to the "sqlurlog - Asynchronous Read Log API" on page 349, and the database log.

Table 23. Fields in the SQLU-RLOG-INFO Structure

Field Name	Data Type	Description
initialLSN	SQLU_LSN	Specifies the LSN value of the first log record that is written after the first database CONNECT statement is issued. For more information, see "Data Structure: SQLU-LSN" on page 357.
firstReadLSN	SQLU_LSN	Specifies the LSN value of the first log record read.
lastReadLSN	SQLU_LSN	Specifies the LSN value of the last log record read.
curActiveLSN	SQLU_LSN	Specifies the LSN value of the current (active) log.
logRecsWritten	sqluint32	Specifies the number of log records written to the buffer.
logBytesWritten	sqluint32	Specifies the number of bytes written to the buffer.

Language Syntax

C Structure

```
typedef SQL_STRUCTURE SQLU_RLOG_INFO
{
SQLU_LSN       initialLSN ;
SQLU_LSN       firstReadLSN ;
SQLU_LSN       lastReadLSN ;
SQLU_LSN       curActiveLSN ;
sqluint32      logRecsWritten ;
sqluint32      logBytesWritten ;
} SQLU_RLOG_INFO;
```

Appendix E. Recovery Sample Programs

Sample Program with No Embedded SQL (backrest.c)

The following sample program shows how to use DB2 backup and restore APIs to:

- Back up a database
- Restore the database
- Rollforward recover the database

For detailed information about the SAMPLE database, see the *SQL Reference*.

The source file for this sample program (`backrest.c`) can be found in the `\sqllib\samples\c` directory on Windows operating systems and OS/2, and in the `/sqllib/samples/c` directory on UNIX based systems. It contains a number of DB2 APIs. The `makefile`, located in the same directory, contains the commands to build this and other sample programs. For general information about creating applications containing DB2 administrative APIs, and detailed information about compile and link options, see the *Application Building Guide*. To build the sample program `backrest` from the source file `backrest.c` on Windows NT or Windows 2000:

1. Copy the files `backrest.c`, `makefile`, `utilapi.c`, and `utilapi.h` to a working directory.
2. If the database manager is not running, issue the **db2start** command from a DB2 command window. To open a CLP-enabled DB2 window, and initialize the DB2 command line environment on the Windows operating system, issue **db2cmd** from a command prompt.
3. Enter `nmake backrest`. The following files are generated:

```
backrest.exe
backrest.ilk
backrest.obj
backrest.pdb
utilapi.obj
```

To run the sample program (executable file), enter:

```
backrest database userID password
```

For example:

```
backrest sample db2admin db2admin
```

The following is an example of the output that this program returns:

Sample Program with No Embedded SQL (backrest.c)

```
This is sample program : backrest.c

NOTE:   Ensure the database is not in use prior to running this program.

Updating the sample database configuration parameter LOGRETAIN
to 'ON' to enable rollforward recovery.

Backing up the 'sample' database.
The database has been successfully backed up.

Updating the sample database configuration parameter LOGRETAIN
to 'OFF'.

Restoring the database 'sample' as 'TESTBACK' (1st pass).
Should get returned value = SQLUD_INACCESSABLE_CONTAINER.
SQLUD_INACCESSABLE_CONTAINER is returned.

Need to SET TABLESPACES CONTAINERS
Tablespace container information for tablespace 1 obtained.
Tablespace containers have been set for tablespace 1.

Restoring the database 'sample' as 'TESTBACK' (2nd pass).
Database sample has been restored as 'TESTBACK'.

The database 'TESTBACK' has been successfully rolled forward.

The database 'TESTBACK' has been successfully dropped.
```

Following is the source listing for the sample program:

```
/*****************************************************************************
**
** Source File Name = backrest.c
**
** Licensed Materials - Property of IBM
**
** (C) COPYRIGHT International Business Machines Corp. 1995, 2000
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
**
**    PURPOSE :
**       an example showing how to use BACKUP & RESTORE APIs in order to:
**          - backup a database
**          - restore a database
**          - roll forward database to return the database to a state it was
**            in prior to the damage...
**
**    APIs USED :
**       BACKUP DATABASE                  sqlubkp()
**       RESTORE DATABASE                 sqlurestore()
**       UPDATE DATABASE CONFIGURATION    sqlfudb()
**       GET TABLESPACE CONTAINER QUERY   sqlbtcq()
**       SET TABLESPACE CONTAINERS        sqlbstsc()
```

```
**       ROLLFORWARD DATABASE              sqluroll()
**
**    STRUCTURES USED :
**       sqlu_tablespace_bkrst_list
**       SQLB_TBSCONTQRY_DATA
**       sqlu_media_list
**       rfwd_input
**       rfwd_output
**       sqlurf_info
**       sqlca
**
**    OTHER FUNCTIONS DECLARED :
**       'C' COMPILER LIBRARY :
**          stdio.h  -  printf
**
**       internal :
**
**       external :
**          check_error :     Checks for SQLCODE error, and prints out any
**          [in UTIL.C]          related information available.
**                               This procedure is located in the UTIL.C file.
**
**    EXTERNAL DEPENDENCIES :
**       - Ensure existence of database for precompile purposes.
**       - Compile and link with the IBM Cset++ compiler (AIX and OS/2)
**         or the Microsoft Visual C++ compiler (Windows)
**         or the compiler supported on your platform.
**
** For more information about these samples see the README file.
**
** For more information on programming in C, see the:
**   -  "Programming in C and C++" section of the Application Development Guide
** For more information on building C applications, see the:
**   -  "Building C Applications" section of the Application Building Guide.
**
** For more information on the SQL language see the SQL Reference.
**
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlutil.h>
#include <sqlenv.h>
#include "utilapi.h"

/* You can change the following path to another directory for which */
/* you have permission. create the "backup" directory in the same path. */

   #define TEMPDIR "."

int main (int argc, char *argv[]) {

   /* Variables for the BACKUP API */
   sqluint32 buff_size;
```

```
sqluint32 num_buff;
struct sqlu_tablespace_bkrst_list *tablespace_list;
struct SQLB_TBSCONTQRY_DATA *cont;
struct sqlca sqlca;
struct sqlu_media_entry media_entry;
struct sqlu_media_list media_list;
char applid[SQLU_APPLID_LEN+1];
char timestamp[SQLU_TIME_STAMP_LEN+1];
char *target_path;

sqluint32 tsc_id;
sqluint32 num_cont;

/* variables for the Update Database Configuration API */
struct sqlfupd itemList;
short log_retain;

/* variables for the ROLLFORWARD API */
struct rfwd_input rfwdInput;
char dbAlias[] = "TESTBACK";
char overFlowLogPath[SQL_PATH_SZ] = TEMPDIR;

struct rfwd_output rfwdOutput;
sqlint32   numReplies;
struct sqlurf_info nodeInfo;

printf ("\nThis is sample program : backrest.c\n\n");

printf ("NOTE:   Ensure the database is not in use prior to running
 this program.\n\n");

if (argc != 4) {
   printf ("\nUSAGE: backrest database userID password\n\n");
   return 1;
} /* endif */

/* Altering the default Database Configuration to enable rollforward
   recovery of the TESTBACK database. */

log_retain = 1;
itemList.token = SQLF_DBTN_LOG_RETAIN;

itemList.ptrvalue = (char *) &log_retain;


printf ("Updating the %s database configuration parameter
 LOGRETAIN \n", argv[1]);
printf ("to 'ON' to enable rollforward recovery.\n\n");

/*****************************************\
* UPDATE DATABASE CONFIGURATION API called *
\*****************************************/
sqlfudb(argv[1], 1, &itemList, &sqlca);
API_SQL_CHECK("updating the database configuration");
```

```
/* Initialize the BACKUP API variables */
 buff_size = 16;
 num_buff = 1;
 tablespace_list = NULL;
 media_list.media_type = 'L';
 media_list.sessions = 1;

 strcpy (media_entry.media_entry, TEMPDIR);

 media_list.target.media = &media_entry;
 printf ("Backing up the '%s' database.\n", argv[1]);
 /*****************\
 * BACKUP DATABASE *
 \*****************/
 sqlubkp (argv[1],
           buff_size,
           SQLUB_OFFLINE,
           SQLUB_FULL,
           SQLUB_BACKUP,
           applid,
           timestamp,
           num_buff,
           tablespace_list,
           &media_list,
           argv[2],
           argv[3],
           NULL,
           0,
           NULL,
           1,
           NULL,
           NULL,
           NULL,
           &sqlca);

 API_SQL_CHECK("backing up the database");
 printf ("The database has been successfully backed up.\n\n");



 /* Altering the default Database Configuration to disable rollforward
    recovery of the TESTBACK database. */

 log_retain = 0;
 itemList.token = SQLF_DBTN_LOG_RETAIN;
 itemList.ptrvalue = (char *) &log_retain;


 printf ("Updating the %s database configuration parameter
  LOGRETAIN \n", argv[1]);
 printf ("to 'OFF'.\n\n");

 /*****************************************\
```

```
* UPDATE DATABASE CONFIGURATION API called *
\*****************************************/
sqlfudb(argv[1], 1, &itemList, &sqlca);
API_SQL_CHECK("updating the database configuration");



/* Initialize the variables for the RESTORE API */
buff_size = 1024;
target_path = NULL;
printf ("Restoring the database '%s' as 'TESTBACK' (1st pass).\n", argv[1]);
printf ("Should get returned value = SQLUD_INACCESSABLE_CONTAINER.\n");
/******************\
* RESTORE DATABASE *
\******************/
sqlurestore(argv[1],
          "TESTBACK",
          buff_size,
          SQLUD_ROLLFWD,
          SQLUD_DATALINK,
          SQLUD_FULL,
          SQLUD_OFFLINE,
          SQLUD_RESTORE_STORDEF,
          applid,
          timestamp,
          target_path,
          num_buff,
          NULL,
          tablespace_list,
          &media_list,
          argv[2],
          argv[3],
          NULL,
          0,
          NULL,
          1,
          NULL,
          NULL,
          NULL,
          &sqlca);
if (sqlca.sqlcode != SQLUD_INACCESSABLE_CONTAINER)
   API_SQL_CHECK("restoring database");

printf ("SQLUD_INACCESSABLE_CONTAINER is returned.\n\n");
printf ("Need to SET TABLESPACES CONTAINERS\n");
/* Set the containers structure to a new list of containers */
tsc_id = 1;
cont = (struct SQLB_TBSCONTQRY_DATA *) malloc
   (sizeof(struct SQLB_TBSCONTQRY_DATA));

/*******************************\
* GET TABLESPACE CONTAINER QUERY *
\*******************************/
sqlbtcq (&sqlca, tsc_id, &num_cont, &cont);
```

```
API_SQL_CHECK("GET TABLESPACE CONTAINER QUERY");
printf ("Tablespace container information for tablespace 1 obtained.\n");
/**************************\
* SET TABLESPACE CONTAINERS *
\**************************/
sqlbstsc (&sqlca,
          SQLB_SET_CONT_INIT_STATE,
          tsc_id,
          num_cont,
          cont);
API_SQL_CHECK("SET TABLESPACE CONTAINERS");
printf ("Tablespace containers have been set for tablespace 1.\n\n");

printf ("Restoring the database '%s' as 'TESTBACK' (2nd pass).\n", argv[1]);
/******************\
* RESTORE DATABASE *
\******************/
sqlurestore (argv[1],
          "TESTBACK",
          buff_size,
          SQLUD_ROLLFWD,
          SQLUD_DATALINK,
          SQLUD_FULL,
          SQLUD_OFFLINE,
          SQLUD_CONTINUE,
          applid,
          timestamp,
          target_path,
          num_buff,
          NULL,
          tablespace_list,
          &media_list,
          argv[2],
          argv[3],
          NULL,
          0,
          NULL,
          1,
          NULL,
          NULL,
          NULL,
          &sqlca);
API_SQL_CHECK("restoring database 2");
printf ("Database %s has been restored as 'TESTBACK'.\n\n", argv[1]);


/**********************\
* ROLLFORWARD DATABASE *
\**********************/
rfwdInput.version=SQLUM_RFWD_VERSION;
rfwdInput.pDbAlias=dbAlias;
rfwdInput.CallerAction=SQLUM_ROLLFWD;
rfwdInput.pStopTime=SQLUM_INFINITY_TIMESTAMP;
rfwdInput.pUserName=argv[2];
rfwdInput.pPassword=argv[3];
```

Sample Program with No Embedded SQL (backrest.c)

```
   rfwdInput.pOverflowLogPath=overFlowLogPath;
   rfwdInput.NumChngLgOvrflw=0;
   rfwdInput.pChngLogOvrflw=NULL;
   rfwdInput.ConnectMode=SQLUM_OFFLINE;
   rfwdInput.pTablespaceList=NULL;
   rfwdInput.AllNodeFlag= SQLURF_ALL_NODES;
   rfwdInput.NumNodes=0;
   rfwdInput.pNodeList=NULL;
   rfwdInput.NumNodeInfo=1;
   rfwdInput.DlMode=0;
   rfwdInput.pReportFile=NULL;
   rfwdInput.pDroppedTblID=NULL;
   rfwdInput.pExportDir=NULL;


   rfwdOutput.pApplicationId=applid;
   rfwdOutput.pNumReplies=&numReplies;
   rfwdOutput.pNodeInfo=&nodeInfo;


   sqluroll( &rfwdInput,
             &rfwdOutput,
             &sqlca);
   API_SQL_CHECK("rolling database forward");
   printf ("The database 'TESTBACK' has been successfully rolled
    forward.\n\n", argv[1]);


   /***************\
   * DROP DATABASE *
   \***************/
   sqledrpd ("TESTBACK",
             &sqlca);
   API_SQL_CHECK("DROP DATABASE");
   printf ("The database 'TESTBACK' has been successfully dropped.\n");


   return 0;
}
```

Sample Program with Embedded SQL (dbrecov.sqc)

The following sample program shows how to use DB2 backup and restore APIs to:

- Back up a database
- Restore the database
- Rollforward recover the database

For detailed information about the SAMPLE database, see the *SQL Reference*.

Note: The `dbrecov` sample files are not installed with DB2 Version 7.2. They are only available for download from the Web.

To download the required files for the dbrecov sample program:

1. Go to http://www.ibm.com/software/data/db2/udb/ad/v7/abg.html.
2. Under "Recent Updates", click on "dbrecov sample files by platform".
3. On the page that appears, choose the link to download the files for your platform. For Windows operating systems and OS/2, the files are contained in a zipped executable file; for UNIX based systems, the files are contained in a tar.gz file.

CAUTION:
Do not run this executable in the directory where your existing DB2 UDB Version 7 sample files reside, because the new utility files will overwrite existing versions of the DB2 UDB Version 7 utility files. The new utility files are not compatible with other sample programs.

The source file for this sample program (dbrecov.sqc) contains both DB2 APIs and embedded SQL calls. For general information about creating applications containing DB2 administrative APIs, and detailed information about compile and link options, see the *Application Building Guide*.

Following are instructions for building and running the dbrecov program on the Windows operating system. To build the program on other supported operating systems, see the README file that is included among the sample files.

1. Run the downloaded executable file, dbrecov_win.exe, in your working directory. This will extract the following files:

```
dbrecov.sqc - Embedded SQL source file for dbrecov program
utilapi.c   - Error-checking utility file for DB2 API programs
utilapi.h   - Header file for utilapi.c
utilemb.sqc - Error-checking utility file for embedded SQL programs
utilemb.h   - Header file for utilemb.sqc
bldmapp.bat - Builds Microsoft Visual C++ application programs
bldvapp.bat - Builds VisualAge C++ application programs
embprep.bat - Precompiles and binds embedded SQL programs
README      - Contains information about the program files
```

2. If the database manager is not running, issue the **db2start** command from a DB2 command window. To open a CLP-enabled DB2 window, and initialize the DB2 command line environment on the Windows operating system, issue **db2cmd** from a command prompt.
3. Enter bldmapp dbrecov, or bldvapp dbrecov, depending on your compiler. The following files are generated:

```
dbrecov.exe
dbrecov.ilk
dbrecov.c
dbrecov.obj
```

```
dbrecov.bnd
dbrecov.pdb
utilemb.c
utilemb.obj
```

To run the sample program (executable file), enter:

```
dbrecov
```

The output will vary depending on your operating system environment. The following is an example of the output that this program returns on a Windows system:

```
THIS SAMPLE SHOWS HOW TO RECOVER A DATABASE.

USE THE DB2 API:
  sqlfxdb -- Get Database Configuration
TO GET THE DATABASE CONFIGURATION AND DETERMINE
THE SERVER WORKING PATH.

NOTE: Backup images will be created on the server
      in the directory D:\DB2,
      and will not be deleted by the program.

**************************************
*** PRUNE THE RECOVERY HISTORY FILE ***
**************************************

USE THE DB2 API:
  db2Prune -- Prune Recovery History File
AND THE SQL STATEMENTS:
  CONNECT
  CONNECT RESET
TO PRUNE THE RECOVERY HISTORY FILE.

  Connecting to 'sample' database...
  Connected to 'sample' database.

  Prune the recovery history file for 'sample' database.

  Disconnecting from 'sample' database...
  Disconnected from 'sample' database.

**************************************
*** BACK UP AND RESTORE A DATABASE ***
**************************************

USE THE DB2 APIs:
  sqlfudb -- Update Database Configuration
  sqlubkp -- Backup Database
  sqlurestore -- Restore Database
TO BACK UP AND RESTORE A DATABASE.

  Update 'sample' database configuration:
    - Disable the database configuration parameter LOGRETAIN
```

```
        i.e., set LOGRETAIN = OFF/NO

  Backing up the 'sample' database...
  Backup finished.
    - backup image size      : 9 MB
    - backup image path      : D:\DB2
    - backup image time stamp: 20010506162032

  Restoring a database ...
    - source image alias     : sample
    - source image time stamp: 20010506162032
    - target database        : sample

-- The following warning report is expected! --
---- warning report ---------------------------

application message = database restore -- start
line                = 506
file                = dbrecov.sqc
SQLCODE             = 2539

SQL2539W  Warning!  Restoring to an existing database that is the same as the
backup image database.  The database files will be deleted.

---- end warning report ----------------------

  Continuing the restore operation...

  Restore finished.

*************************
*** REDIRECTED RESTORE ***
*************************

USE THE DB2 APIs:
  sqlfudb -- Update Database Configuration
  sqlubkp -- Backup Database
  sqlecrea -- Create Database
  sqlurestore -- Restore Database
  sqlbmtsq -- Tablespace Query
  sqlbtcq -- Tablespace Container Query
  sqlbstsc -- Set Tablespace Containers
  sqlefmem -- Free Memory
  sqledrpd -- Drop Database
TO BACK UP AND DO A REDIRECTED RESTORE OF A DATABASE.

  Update 'sample' database configuration:
    - Disable the database configuration parameter LOGRETAIN
        i.e., set LOGRETAIN = OFF/NO

  Backing up the 'sample' database...
  Backup finished.
    - backup image size      : 9 MB
    - backup image path      : D:\DB2
    - backup image time stamp: 20010506162125
```

```
  Restoring a database ...
    - source image alias     : sample
    - source image time stamp: 20010506162125
    - target database        : RRDB

-- The following warning report is expected! --
---- warning report ---------------------------

application message = database restore -- start
line               = 776
file               = dbrecov.sqc
SQLCODE            = 1277

SQL1277N  Restore has detected that one or more table space containers are
inaccessible, or has set their state to 'storage must be defined'.

---- end warning report ----------------------

  Continuing the restore operation...

  Find and redefine inaccessible containers.

    Redefine inaccessible container:
      - table space name: SYSCATSPACE
      - default container name: D:\DB2\NODE0000\SQL00003\SQLT0000.0
      - container type: path
      - new container name: D:\DB2\SQLT0000.0

    Redefine inaccessible container:
      - table space name: TEMPSPACE1
      - default container name: D:\DB2\NODE0000\SQL00003\SQLT0001.0
      - container type: path
      - new container name: D:\DB2\SQLT0001.0

    Redefine inaccessible container:
      - table space name: USERSPACE1
      - default container name: D:\DB2\NODE0000\SQL00003\SQLT0002.0
      - container type: path
      - new container name: D:\DB2\SQLT0002.0

  Restore finished.

  Drop the 'RRDB' database.

****************************
*** ROLLFORWARD RECOVERY ***
****************************

USE THE DB2 APIs:
  sqlfudb -- Update Database Configuration
  sqlubkp -- Backup Database
  sqlecrea -- Create Database
  sqlurestore -- Restore Database
  sqluroll -- Rollforward Database
```

```
  sqledrpd -- Drop Database
TO BACK UP, RESTORE, AND ROLL A DATABASE FORWARD.

  Update 'sample' database configuration:
    - Enable the configuration parameter LOGRETAIN
        i.e., set LOGRETAIN = RECOVERY/YES

  Backing up the 'sample' database...
  Backup finished.
    - backup image size      : 9 MB
    - backup image path      : D:\DB2
    - backup image time stamp: 20010506162223

  Restoring a database ...
    - source image alias     : sample
    - source image time stamp: 20010506162223
    - target database        : RFDB

  Restore finished.

  Rolling 'RFDB' database forward ...
  Rollforward finished.

  Drop the 'RFDB' database.

*****************************
*** ASYNCHRONOUS READ LOG ***
*****************************

USE THE DB2 APIs:
  sqlfudb -- Update Database Configuration
  sqlubkp -- Backup Database
  sqlurlog -- Asynchronous Read Log
AND THE SQL STATEMENTS:
  CONNECT
  ALTER TABLE
  COMMIT
  INSERT
  DELETE
  ROLLBACK
  CONNECT RESET
TO READ LOG RECORDS FOR THE CURRENT CONNECTION.

  Update 'sample' database configuration:
    - Enable the database configuration parameter LOGRETAIN
        i.e., set LOGRETAIN = RECOVERY/YES

  Backing up the 'sample' database...
  Backup finished.
    - backup image size      : 9 MB
    - backup image path      : D:\DB2
    - backup image time stamp: 20010506162247

  Connecting to 'sample' database...
  Connected to 'sample' database.
```

```
  Invoke the following SQL statements:
    ALTER TABLE emp_resume DATA CAPTURE CHANGES;
    COMMIT;
    INSERT INTO emp_resume
      VALUES('000777', 'ascii', 'This is a new resume.');
            ('777777', 'ascii', 'This is another new resume');
    COMMIT;
    DELETE FROM emp_resume WHERE empno = '000777';
    DELETE FROM emp_resume WHERE empno = '777777';
    COMMIT;
    DELETE FROM emp_resume WHERE empno = '000140';
    ROLLBACK;
    ALTER TABLE emp_resume DATA CAPTURE NONE;
    COMMIT;

  Start reading database log.

-- The following warning report is expected! --
---- warning report --------------------------

application message = database log info -- get
line               = 1457
file               = dbrecov.sqc
SQLCODE            = 2653

SQL2653W  A Restore, Forward or Crash Recovery may have reused log sequence
number ranges.  Reason code "1".

---- end warning report ----------------------

    Record type: Normal
      component ID: DMS log record
      function ID:  Alter Table Attribute
        Propagation attribute is changed to: ON

    Record type: Normal
      component ID: DMS log record
      function ID: Update Record
        oldRID: 2
        old subrecord length: 76
        old subrecord offset: 0
        subrecord type: Updatable, Internal control
        newRID: 2
        new subrecord length: 76
        new subrecord offset: 2916
        subrecord type: Updatable, Internal control

    Record type: Local pending list
      UTC transaction committed (in seconds since 01/01/70): 989180591
      authorization ID of the application: MELNYK

    Record type: Normal
      component ID: DMS log record
      function ID: Insert Record
```

```
    RID: 12
    subrecord length: 88
    subrecord offset: 486
    subrecord type: Updatable, Formatted user data
    user data fixed length: 15
    user data:
    30 30 30 37 37 37 0F 00 05 00 *000777....*
    14 00 3C 00 00 61 73 63 69 69 *.....ascii*
    49 06 01 00 00 00 00 00 15 00 *I.........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 3C 00 01 00 00 00 15 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*

Record type: Normal
  component ID: DMS log record
  function ID: Insert Record
    RID: 13
    subrecord length: 88
    subrecord offset: 398
    subrecord type: Updatable, Formatted user data
    user data fixed length: 15
    user data:
    37 37 37 37 37 37 0F 00 05 00 *777777....*
    14 00 3C 00 00 61 73 63 69 69 *.....ascii*
    49 06 01 00 00 00 00 00 1A 00 *I.........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 3C 00 01 00 00 00 1A 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 01 00 00 00 *..........*

Record type: Normal commit
  UTC transaction committed (in seconds since 01/01/70): 989180591
  authorization ID of the application: MELNYK

Record type: Normal
  component ID: DMS log record
  function ID: Delete Record
    RID: 12
    subrecord length: 88
    subrecord offset: 0
    subrecord type: Updatable, Formatted user data
    user data fixed length: 15
    user data:
    30 30 30 37 37 37 0F 00 05 00 *000777....*
    14 00 3C 00 00 61 73 63 69 69 *.....ascii*
    49 06 01 00 00 00 00 00 15 00 *I.........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 3C 00 01 00 00 00 15 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
```

```
Record type: Normal
  component ID: DMS log record
  function ID: Delete Record
    RID: 13
    subrecord length: 88
    subrecord offset: 0
    subrecord type: Updatable, Formatted user data
    user data fixed length: 15
    user data:
    37 37 37 37 37 37 0F 00 05 00 *777777....*
    14 00 3C 00 00 61 73 63 69 69 *.....ascii*
    49 06 01 00 00 00 00 00 1A 00 *I.........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 3C 00 01 00 00 00 1A 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 01 00 00 00 *..........*

Record type: Normal commit
  UTC transaction committed (in seconds since 01/01/70): 989180591
  authorization ID of the application: MELNYK

Record type: Normal
  component ID: DMS log record
  function ID: Delete Record
    RID: 6
    subrecord length: 88
    subrecord offset: 0
    subrecord type: Updatable, Formatted user data
    user data fixed length: 15
    user data:
    30 30 30 31 34 30 0F 00 05 00 *000140....*
    14 00 3C 00 00 61 73 63 69 69 *.....ascii*
    49 06 01 00 00 00 00 00 24 05 *I.........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 01 3C 00 02 00 00 00 24 05 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 00 00 00 00 *..........*
    00 00 00 00 00 00 14 00 00 00 *..........*

Record type: Normal
  component ID: DMS log record
  function ID: Delete Record
    RID: 7
    subrecord length: 89
    subrecord offset: 0
    subrecord type: Updatable, Formatted user data
    user data fixed length: 15
    user data:
    30 30 30 31 34 30 0F 00 06 00 *000140....*
    15 00 3C 00 00 73 63 72 69 70 *.....scrip*
    74 49 06 01 00 00 00 00 00 56 *tI.......V*
    07 00 00 00 00 00 00 00 00 00 *..........*
    00 00 01 3C 00 02 00 00 00 56 *.........V*
    07 00 00 00 00 00 00 00 00 00 *..........*
```

```
      00 00 00 00 00 00 00 00 00 00 *..........*
      00 00 00 00 00 00 00 16 00 00 *..........*
      00                            *.         *

    Record type: Compensation
      component ID: DMS log record
      function ID: Undo Delete Record
        RID: 7
        subrecord length: 89
        subrecord offset: 397
        subrecord type: Updatable, Formatted user data
        user data fixed length: 15
        user data:
        30 30 30 31 34 30 0F 00 06 00 *000140....*
        15 00 3C 00 00 73 63 72 69 70 *.....scrip*
        74 49 06 01 00 00 00 00 00 56 *tI.......V*
        07 00 00 00 00 00 00 00 00 00 *..........*
        00 00 01 3C 00 02 00 00 00 56 *.........V*
        07 00 00 00 00 00 00 00 00 00 *..........*
        00 00 00 00 00 00 00 00 00 00 *..........*
        00 00 00 00 00 00 00 16 00 00 *..........*
        00                            *.         *

    Record type: Compensation
      component ID: DMS log record
      function ID: Undo Delete Record
        RID: 6
        subrecord length: 88
        subrecord offset: 309
        subrecord type: Updatable, Formatted user data
        user data fixed length: 15
        user data:
        30 30 30 31 34 30 0F 00 05 00 *000140....*
        14 00 3C 00 00 61 73 63 69 69 *.....ascii*
        49 06 01 00 00 00 00 00 24 05 *I.........*
        00 00 00 00 00 00 00 00 00 00 *..........*
        00 01 3C 00 02 00 00 00 24 05 *..........*
        00 00 00 00 00 00 00 00 00 00 *..........*
        00 00 00 00 00 00 00 00 00 00 *..........*
        00 00 00 00 00 00 14 00 00 00 *..........*

    Record type: Normal abort
      authorization ID of the application: MELNYK

    Record type: Normal
      component ID: DMS log record
      function ID:  Alter Table Attribute
        Propagation attribute is changed to: OFF

    Record type: Local pending list
      UTC transaction committed (in seconds since 01/01/70): 989180591
      authorization ID of the application: MELNYK

  Disconnecting from 'sample' database...
  Disconnected from 'sample' database.
```

```
*********************************************
*** READ A DATABASE RECOVERY HISTORY FILE ***
*********************************************

USE THE DB2 APIs:
  db2HistoryOpenScan -- Open Recovery History File Scan
  db2HistoryGetEntry -- Get Next Recovery History File Entry
  db2HistoryCloseScan -- Close Recovery History File Scan
TO READ A DATABASE RECOVERY HISTORY FILE.

  Open recovery history file for 'sample' database.

  Read entry number 0.

  Display entry number 0.
    object part: 20010506162032001
    end time: 200105061620
    first log: S0000000
    last log: S0000000
    ID:
    table qualifier:
    table name:
    location: D:\DB2\SAMPLE.0\DB2\NODE0000\CATN0000\20010506
    comment: DB2 BACKUP SAMPLE OFFLINE
    command text:
    history file entry ID: 48
    table spaces:
      SYSCATSPACE
      USERSPACE1
    type of operation: B
    granularity of the operation: D
    operation type: F
    entry status: A
    device type: D
    SQLCA:
      sqlcode: 0
      sqlstate:
      message:

  Read entry number 1.

  Display entry number 1.
    object part: 20010506162058001
    end time: 200105061621
    first log: S0000000
    last log: S0000000
    ID: 20010506162032
    table qualifier:
    table name:
    location: D:\DB2\SAMPLE.0\DB2\NODE0000\CATN0000\20010506
    comment: DB2 RESTORE SAMPLE NO RF
    command text:
    history file entry ID: 49
    table spaces:
```

```
    SYSCATSPACE
    USERSPACE1
  type of operation: R
  granularity of the operation: D
  operation type: F
  entry status: A
  device type: D
  SQLCA:
    sqlcode: 0
    sqlstate:
    message:

Read entry number 2.

Display entry number 2.
  object part: 20010506162125001
  end time: 200105061622
  first log: S0000000
  last log: S0000000
  ID:
  table qualifier:
  table name:
  location: D:\DB2\SAMPLE.0\DB2\NODE0000\CATN0000\20010506
  comment: DB2 BACKUP SAMPLE OFFLINE
  command text:
  history file entry ID: 50
  table spaces:
    SYSCATSPACE
    USERSPACE1
  type of operation: B
  granularity of the operation: D
  operation type: F
  entry status: A
  device type: D
  SQLCA:
    sqlcode: 0
    sqlstate:
    message:

Read entry number 3.

Display entry number 3.
  object part: 20010506162223001
  end time: 200105061622
  first log: S0000000
  last log: S0000000
  ID:
  table qualifier:
  table name:
  location: D:\DB2\SAMPLE.0\DB2\NODE0000\CATN0000\20010506
  comment: DB2 BACKUP SAMPLE OFFLINE
  command text:
  history file entry ID: 51
  table spaces:
    SYSCATSPACE
```

```
      USERSPACE1
    type of operation: B
    granularity of the operation: D
    operation type: F
    entry status: A
    device type: D
    SQLCA:
      sqlcode: 0
      sqlstate:
      message:

  Read entry number 4.

  Display entry number 4.
    object part: 20010506162247001
    end time: 200105061623
    first log: S0000000
    last log: S0000000
    ID:
    table qualifier:
    table name:
    location: D:\DB2\SAMPLE.0\DB2\NODE0000\CATN0000\20010506
    comment: DB2 BACKUP SAMPLE OFFLINE
    command text:
    history file entry ID: 52
    table spaces:
      SYSCATSPACE
      USERSPACE1
    type of operation: B
    granularity of the operation: D
    operation type: F
    entry status: A
    device type: D
    SQLCA:
      sqlcode: 0
      sqlstate:
      message:

  Close recovery history file for 'sample' database.

*****************************************************
*** UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY ***
*****************************************************

USE THE DB2 APIs:
  db2HistoryOpenScan -- Open Recovery History File Scan
  db2HistoryGetEntry -- Get Next Recovery History File Entry
  db2HistoryUpdate -- Update Recovery History File
  db2HistoryCloseScan -- Close Recovery History File Scan
TO UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY.

  Open the recovery history file for 'sample' database.

  Read the first entry in the recovery history file.
```

```
Display the first entry.
  object part: 20010506162032001
  end time: 200105061620
  first log: S0000000
  last log: S0000000
  ID:
  table qualifier:
  table name:
  location: D:\DB2\SAMPLE.0\DB2\NODE0000\CATN0000\20010506
  comment: DB2 BACKUP SAMPLE OFFLINE
  command text:
  history file entry ID: 48
  table spaces:
    SYSCATSPACE
    USERSPACE1
  type of operation: B
  granularity of the operation: D
  operation type: F
  entry status: A
  device type: D
  SQLCA:
    sqlcode: 0
    sqlstate:
    message:

Connecting to 'sample' database...
Connected to 'sample' database.

Update the first entry in the history file:
  new location = 'this is the NEW LOCATION'
  new comment = 'this is the NEW COMMENT'

Disconnecting from 'sample' database...
Disconnected from 'sample' database.

Close recovery history file for 'sample' database.

Read the first recovery history file entry.

Display the first entry.
  object part: 20010506162032001
  end time: 200105061620
  first log: S0000000
  last log: S0000000
  ID:
  table qualifier:
  table name:
  location: this is the NEW LOCATION
  comment: this is the NEW COMMENT
  command text:
  history file entry ID: 48
  table spaces:
    SYSCATSPACE
    USERSPACE1
  type of operation: B
```

```
    granularity of the operation: D
    operation type: F
    entry status: A
    device type: D
    SQLCA:
      sqlcode: 0
      sqlstate:
      message:

  Close the recovery history file for 'sample' database.

**************************************
*** PRUNE THE RECOVERY HISTORY FILE ***
**************************************

USE THE DB2 API:
  db2Prune -- Prune Recovery History File
AND THE SQL STATEMENTS:
  CONNECT
  CONNECT RESET
TO PRUNE THE RECOVERY HISTORY FILE.

  Connecting to 'sample' database...
  Connected to 'sample' database.

  Prune the recovery history file for 'sample' database.

  Disconnecting from 'sample' database...
  Disconnected from 'sample' database.
```

Following is the source listing for the sample program:

```
/*****************************************************************************
**
** Source File Name: dbrecov.sqc
**
** Licensed Materials - Property of IBM
**
** (C) COPYRIGHT International Business Machines Corp. 2001
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
**
**
**    PURPOSE:
**    This sample shows how to recover a database.
**
**    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
**        db2Prune -- Prune Recovery History File
**        sqlbmtsq -- Tablespace Query
```

```
**        sqlbstsc -- Set Tablespace Containers
**        sqlbtcq -- Tablespace Container Query
**        sqledrpd -- Drop Database
**        sqlefmem -- Free Memory
**        sqlfudb -- Update Database Configuration
**        sqlfxdb -- Get Database Configuration
**        sqlubkp -- Backup Database
**        sqlurestore -- Restore Database
**        sqlurlog -- Asynchronous Read Log
**        sqluroll -- Rollforward Database
**
** For detailed information about database backup and recovery, see the
** "Data Recovery and High Availability Guide and Reference". This manual will
** help you to determine which database and table space recovery methods are
** best suited to your business environment.
**
** For more information about the sample programs, see the README file.
**
** For more information about programming in C, see the
** "Programming in C and C++" section of the "Application Development Guide".
**
** For more information about building C applications, see the
** section for your compiler in the "Building Applications" chapter
** for your platform in the "Application Building Guide".
**
** For more information about SQL, see the "SQL Reference".
**
** For more information about DB2 APIs, see the "Administrative API Reference".
**
** For the latest information on programming, compiling, and running DB2
** applications, visit the DB2 application development Web site at
**     http://www.software.ibm.com/data/db2/udb/ad
**
******************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#include <db2ApiDf.h>
#include "utilemb.h"

int DbRecoveryHistoryFilePrune(char *, char *, char *);
int DbBackupAndRestore(char *, char *, char *, char *, char *);
int DbBackupAndRedirectedRestore(char *, char *, char *, char *, char *);
int DbBackupRestoreAndRollforward(char *, char *, char *, char *, char *);
int DbLogRecordsForCurrentConnectionRead(char *, char *, char *, char *);
int DbRecoveryHistoryFileRead(char *);
int DbFirstRecoveryHistoryFileEntryUpdate(char *, char *, char *);


/* support function called by the main() */
int ServerWorkingPathGet(char *, char *);

/* support function called by DbBackupAndRedirectedRestore() */
```

```
int InaccessableContainersRedefine(char *);

/* support function called by DbBackupAndRedirectedRestore() and
   DbBackupRestoreAndRollforward() */
int DbDrop(char *);

/* support function called by DbLogRecordsForCurrentConnectionRead() */
int LogBufferDisplay(char *, sqluint32);
int LogRecordDisplay(char *, sqluint32, sqluint16, sqluint16);
int SimpleLogRecordDisplay(sqluint16, sqluint16, char *, sqluint32);
int ComplexLogRecordDisplay(sqluint16, sqluint16, char *, sqluint32,
                            sqluint8, char *, sqluint32);
int LogSubRecordDisplay(char *, sqluint16);
int UserDataDisplay(char *, sqluint16);

/* support functions called by DbRecoveryHistoryFileRead() and
   DbFirstRecoveryHistoryFileEntryUpdate() */
int HistoryEntryDataFieldsAlloc(struct db2HistoryData *);
int HistoryEntryDisplay(struct db2HistoryData);
int HistoryEntryDataFieldsFree(struct db2HistoryData *);

/* DbCreate will create a new database on the server with the server's
    code page.
   Use this function only if you want to restore a remote database.
   This support function is being called by DbBackupAndRedirectedRestore()
   and DbBackupRestoreAndRollforward().  */
int DbCreate(char *, char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1];
  char serverWorkingPath[SQL_PATH_SZ + 1];
  char restoredDbAlias[SQL_ALIAS_SZ + 1];
  char redirectedRestoredDbAlias[SQL_ALIAS_SZ + 1];
  char rolledForwardDbAlias[SQL_ALIAS_SZ + 1];
  sqluint16 savedLogRetainValue;
  char dbAlias[SQL_ALIAS_SZ + 1];
  char user[USERID_SZ + 1];
  char pswd[PSWD_SZ + 1];

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  printf("\nTHIS SAMPLE SHOWS HOW TO RECOVER A DATABASE.\n");

  strcpy(restoredDbAlias, dbAlias);
  strcpy(redirectedRestoredDbAlias, "RRDB");
  strcpy(rolledForwardDbAlias, "RFDB");

  /* attach to a local or remote instance */
```

```
rc = InstanceAttach(nodeName, user, pswd);
if (rc != 0)
{
  return rc;
}

printf("\nUSE THE DB2 API:\n");
printf("  sqlfxdb -- Get Database Configuration\n");
printf("TO GET THE DATABASE CONFIGURATION AND DETERMINE\n");
printf("THE SERVER WORKING PATH.\n");

/* get the server working path */
rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
if (rc != 0)
{
  return rc;
}

printf("\nNOTE: Backup images will be created on the server\n");
printf("      in the directory %s,\n", serverWorkingPath);
printf("      and will not be deleted by the program.\n");

/* call the sample functions */
rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);

rc = DbBackupAndRestore(dbAlias,
                        restoredDbAlias,
                        user,
                        pswd,
                        serverWorkingPath);

rc = DbBackupAndRedirectedRestore(dbAlias,
                                  redirectedRestoredDbAlias,
                                  user,
                                  pswd,
                                  serverWorkingPath);

rc = DbBackupRestoreAndRollforward(dbAlias,
                                   rolledForwardDbAlias,
                                   user,
                                   pswd,
                                   serverWorkingPath);

rc = DbLogRecordsForCurrentConnectionRead(dbAlias,
                                          user,
                                          pswd,
                                          serverWorkingPath);

rc = DbRecoveryHistoryFileRead(dbAlias);

rc = DbFirstRecoveryHistoryFileEntryUpdate(dbAlias, user, pswd);

rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);

/* detach from the local or remote instance */
```

```
  rc = InstanceDetach(nodeName);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* end main */

int ServerWorkingPathGet(char dbAlias[], char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqlfupd dbConfigFields[1];
  char serverLogPath[SQL_PATH_SZ + 1];
  int len;

  /* get the server log path */
  /* SQLF_DBTN_LOGPATH is a token of the non-updatable database configuration
     parameter 'logpath'; it is used to get the server log path */
  dbConfigFields[0].token = SQLF_DBTN_LOGPATH;
  dbConfigFields[0].ptrvalue =
    (char *)malloc((SQL_PATH_SZ + 1) * sizeof(char));

  /* get database configuration */
  /* the API sqlfxdb returns the values of individual entries in a
     database configuration file */
  sqlfxdb(dbAlias, 1, &dbConfigFields[0], &sqlca);
  DB2_API_CHECK("server log path -- get");

  strcpy(serverLogPath, dbConfigFields[0].ptrvalue);
  free(dbConfigFields[0].ptrvalue);

  /* choose the server working path; if, for example, serverLogPath =
     "C:\DB2\NODE0001\....", we'll keep "C:\DB2" for the serverWorkingPath
     variable; backup images created in this sample will be placed under
     the 'serverWorkingPath' directory */
  len = (int)(strstr(serverLogPath, "NODE") - serverLogPath - 1);
  memcpy(serverWorkingPath, serverLogPath, len);
  serverWorkingPath[len] = '\0';

  return 0;
} /* ServerWorkingPathGet */

int DbCreate(char existingDbAlias[], char newDbAlias[])
{
  struct sqlca sqlca;
  char dbName[SQL_DBNAME_SZ + 1];
  char dbLocalAlias[SQL_ALIAS_SZ + 1];
  char dbPath[SQL_PATH_SZ + 1];
  struct sqledbdesc dbDescriptor;
  struct sqledbcountryinfo countryInfo;
  struct sqlfupd dbConfigFields[2];

  printf("\n  Create '%s' empty database with the same code set as
```

```
   '%s' database.\n",
         newDbAlias, existingDbAlias);

  /* initialize dbConfigFields */
  dbConfigFields[0].token = SQLF_DBTN_TERRITORY;
  dbConfigFields[0].ptrvalue = (char *)malloc(10 * sizeof(char));
  memset(dbConfigFields[0].ptrvalue, '\0', 10);
  dbConfigFields[1].token = SQLF_DBTN_CODESET;
  dbConfigFields[1].ptrvalue = (char *)malloc(20 * sizeof(char));
  memset(dbConfigFields[1].ptrvalue, '\0', 20);

  /* get two database configuration fields */
  sqlfxdb(existingDbAlias, 2, &dbConfigFields[0], &sqlca);
  DB2_API_CHECK("DB Config. -- Get");

  /* create a new database */
  strcpy(dbName, newDbAlias);
  strcpy(dbLocalAlias, newDbAlias);
  strcpy(dbPath, "");

  strcpy(dbDescriptor.sqldbdid, SQLE_DBDESC_2);
  dbDescriptor.sqldbccp = 0;
  dbDescriptor.sqldbcss = SQL_CS_NONE;

  strcpy(dbDescriptor.sqldbcmt, "");
  dbDescriptor.sqldbsgp = 0;
  dbDescriptor.sqldbnsg = 10;
  dbDescriptor.sqltsext = -1;
  dbDescriptor.sqlcatts = NULL;
  dbDescriptor.sqlusrts = NULL;
  dbDescriptor.sqltmpts = NULL;

  strcpy(countryInfo.sqldbcodeset, (char *)dbConfigFields[1].ptrvalue);
  strcpy(countryInfo.sqldblocale, (char *)dbConfigFields[0].ptrvalue);

  /* create database */
  sqlecrea(dbName,
           dbLocalAlias,
           dbPath,
           &dbDescriptor,
           &countryInfo,
           '\0',
           NULL,
           &sqlca);
  DB2_API_CHECK("Database -- Create");

  /* free the allocated memory */
  free(dbConfigFields[0].ptrvalue);
  free(dbConfigFields[1].ptrvalue);

  return 0;
} /* DbCreate */

int DbDrop(char dbAlias[])
{
```

```
  struct sqlca sqlca;

  printf("\n  Drop the '%s' database.\n", dbAlias);

  /* drop and uncatalog the database */
  sqledrpd(dbAlias, &sqlca);
  DB2_API_CHECK("Database -- Drop");

  return 0;
} /* DbDrop */

int DbBackupAndRestore(char dbAlias[],
                       char restoredDbAlias[],
                       char user[],
                       char pswd[],
                       char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqlfupd dbCfgParameters[1];
  unsigned short logretain;
  sqluint32 backupBufferSize;
  sqluint32 backupMode;
  sqluint32 backupType;
  sqluint32 backupCallerAction;
  char backupAppId[SQLU_APPLID_LEN + 1];
  char backupStartTimestamp[SQLU_TIME_STAMP_LEN + 1];
  sqluint32 backupBuffersNb;
  struct sqlu_tablespace_bkrst_list backupTablespaceList;
  struct sqlu_media_list backupTargetMediaList;
  struct sqlu_media_entry backupTargetMediaEntries[1];
  sqluint32 backupParallelismDegree;
  sqluint32 backupImageSize;
  sqluint32 restoreBufferSize;
  sqluint32 rollforwardPendingState;
  sqluint32 datalinkMode;
  sqluint32 restoreType;
  sqluint32 restoreMode;
  sqluint32 restoreCallerAction;
  char restoreAppId[SQLU_APPLID_LEN + 1];
  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];
  char *restoreTargetPath;
  sqluint32 restoreBuffersNb;
  struct sqlu_tablespace_bkrst_list restoreTablespaceList;
  struct sqlu_media_list restoreSourceMediaList;
  struct sqlu_media_entry restoreSourceMediaEntries[1];
  sqluint32 restoreParallelismDegree;

  printf("\n************************************\n");
  printf("*** BACK UP AND RESTORE A DATABASE ***\n");
  printf("************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  sqlfudb -- Update Database Configuration\n");
  printf("  sqlubkp -- Backup Database\n");
  printf("  sqlurestore -- 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");

/* SQLF_DBTN_LOG_RETAIN is a token of the updatable database configuration
   parameter 'logretain'; it is used to update the database configuration
   file */
dbCfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
dbCfgParameters[0].ptrvalue = (char *)&logretain;

/* disable the database configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_DISABLE;

/* The API sqlfudb is used to modify individual entries in a specific
   database configuration file. */
sqlfudb(dbAlias, 1, dbCfgParameters, &sqlca);
DB2_API_CHECK("Db Log Retain -- Disable");

/*******************************/
/*    BACK UP THE DATABASE    */
/*******************************/
printf("\n  Backing up the '%s' database...\n", dbAlias);

backupBufferSize = 16; /*  16 x 4KB */
backupMode = SQLUB_OFFLINE;
backupType = SQLUB_FULL;
backupBuffersNb = 1;
backupTablespaceList.num_entry = 0; /* number of table spaces */
backupTablespaceList.tablespace = NULL;
backupTargetMediaList.media_type = SQLU_LOCAL_MEDIA;
backupTargetMediaList.sessions = 1; /* number of elements in the target */
backupTargetMediaList.target.media = backupTargetMediaEntries;
strcpy(backupTargetMediaEntries[0].media_entry, serverWorkingPath);
backupParallelismDegree = 1;

backupCallerAction = SQLUB_BACKUP;

/* The API sqlubkp creates a backup copy of a database.
   This API automatically establishes a connection to the specified
    database.
   (This API can also be used to create a backup copy of a table space). */
sqlubkp(dbAlias,
        backupBufferSize,
        backupMode,
        backupType,
        backupCallerAction,
        backupAppId,
        backupStartTimestamp,
        backupBuffersNb,
        &backupTablespaceList,
        &backupTargetMediaList,
        user,
        pswd,
```

```
        NULL,
        0,
        NULL,
        backupParallelismDegree,
        &backupImageSize,
        NULL,
        NULL,
        &sqlca);
DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */

  /* depending on the sqlca.sqlcode value, user action may be */
  /* required, such as mounting a new tape */

  printf("\n  Continuing the backup operation...\n");

  backupCallerAction = SQLUB_CONTINUE;

  /* back up the database */
  sqlubkp(dbAlias,
          backupBufferSize,
          backupMode,
          backupType,
          backupCallerAction,
          backupAppId,
          backupStartTimestamp,
          backupBuffersNb,
          &backupTablespaceList,
          &backupTargetMediaList,
          user,
          pswd,
          NULL,
          0,
          NULL,
          backupParallelismDegree,
          &backupImageSize,
          NULL,
          NULL,
          &sqlca);
  DB2_API_CHECK("Database -- Backup");
}

printf("  Backup finished.\n");
printf("    - backup image size      : %d MB\n", backupImageSize);
printf("    - backup image path      : %s\n",
       backupTargetMediaEntries[0].media_entry);
printf("    - backup image time stamp: %s\n",
       backupStartTimestamp);

/*****************************/
/*    RESTORE THE DATABASE    */
/*****************************/
```

```
  restoreBufferSize = 1024; /*  1024 x 4KB */
  rollforwardPendingState = SQLUD_NOROLLFWD;
  datalinkMode = SQLUD_NODATALINK;
  restoreType = SQLUD_FULL;
  restoreMode = SQLUD_OFFLINE;
  strcpy(restoreTimestamp, backupStartTimestamp);
  restoreTargetPath = NULL;
  restoreBuffersNb = 1;
  restoreTablespaceList.num_entry = 0; /* number of table spaces */
  restoreTablespaceList.tablespace = NULL;
  restoreSourceMediaList.media_type = SQLU_LOCAL_MEDIA;
  restoreSourceMediaList.sessions = 1; /* number of elements in the target */
  restoreSourceMediaList.target.media = restoreSourceMediaEntries;
  strcpy(restoreSourceMediaEntries[0].media_entry, serverWorkingPath);
  restoreParallelismDegree = 1;

  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);

  restoreCallerAction = SQLUD_RESTORE;

  /* The API sqlurestore is used to restore a database that has been backed
     up using the API sqlubkp. */
  sqlurestore(dbAlias,
              restoredDbAlias,
              restoreBufferSize,
              rollforwardPendingState,
              datalinkMode,
              restoreType,
              restoreMode,
              restoreCallerAction,
              restoreAppId,
              restoreTimestamp,
              restoreTargetPath,
              restoreBuffersNb,
              NULL,
              &restoreTablespaceList,
              &restoreSourceMediaList,
              user,
              pswd,
              NULL,
              0,
              NULL,
              restoreParallelismDegree,
              NULL,
              NULL,
              NULL,
              &sqlca);
/*  DB2_API_CHECK("database restore -- start");  */
  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 */

    restoreCallerAction = SQLUD_CONTINUE;

    /* restore the database */
    sqlurestore(dbAlias,
                restoredDbAlias,
                restoreBufferSize,
                rollforwardPendingState,
                datalinkMode,
                restoreType,
                restoreMode,
                restoreCallerAction,
                restoreAppId,
                restoreTimestamp,
                restoreTargetPath,
                restoreBuffersNb,
                NULL,
                &restoreTablespaceList,
                &restoreSourceMediaList,
                user,
                pswd,
                NULL,
                0,
                NULL,
                restoreParallelismDegree,
                NULL,
                NULL,
                NULL,
                &sqlca);
    DB2_API_CHECK("database restore -- continue");
  }

  printf("\n  Restore finished.\n");

  return 0;
} /* DbBackupAndRestore */

int DbBackupAndRedirectedRestore(char dbAlias[],
                                 char restoredDbAlias[],
                                 char user[],
                                 char pswd[],
                                 char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqlfupd dbCfgParameters[1];
  unsigned short logretain;
  sqluint32 backupBufferSize;
  sqluint32 backupMode;
```

```
sqluint32 backupType;
sqluint32 backupCallerAction;
char backupAppId[SQLU_APPLID_LEN + 1];
char backupStartTimestamp[SQLU_TIME_STAMP_LEN + 1];
sqluint32 backupBuffersNb;
struct sqlu_tablespace_bkrst_list backupTablespaceList;
struct sqlu_media_list backupTargetMediaList;
struct sqlu_media_entry backupTargetMediaEntries[1];
sqluint32 backupParallelismDegree;
sqluint32 backupImageSize;
sqluint32 restoreBufferSize;
sqluint32 rollforwardPendingState;
sqluint32 datalinkMode;
sqluint32 restoreType;
sqluint32 restoreMode;
sqluint32 restoreCallerAction;
char restoreAppId[SQLU_APPLID_LEN + 1];
char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];
char *restoreTargetPath;
sqluint32 restoreBuffersNb;
struct sqlu_tablespace_bkrst_list restoreTablespaceList;
struct sqlu_media_list restoreSourceMediaList;
struct sqlu_media_entry restoreSourceMediaEntries[1];
sqluint32 restoreParallelismDegree;

printf("\n*************************\n");
printf("*** REDIRECTED RESTORE ***\n");
printf("*************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  sqlfudb -- Update Database Configuration\n");
printf("  sqlubkp -- Backup Database\n");
printf("  sqlecrea -- Create Database\n");
printf("  sqlurestore -- 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");

/* SQLF_DBTN_LOG_RETAIN is a token of the updatable database configuration
   parameter 'logretain'; it is used to update the database configuration
   file */
dbCfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
dbCfgParameters[0].ptrvalue = (char *)&logretain;

/* disable the database configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_DISABLE;

/* The API sqlfudb is used to modify individual entries in a specific
   database configuration file. */
```

```
sqlfudb(dbAlias, 1, dbCfgParameters, &sqlca);
DB2_API_CHECK("Db Log Retain -- Disable");

/*****************************/
/*    BACK UP THE DATABASE    */
/*****************************/
printf("\n  Backing up the '%s' database...\n", dbAlias);

backupBufferSize = 16; /*  16 x 4KB */
backupMode = SQLUB_OFFLINE;
backupType = SQLUB_FULL;
backupBuffersNb = 1;
backupTablespaceList.num_entry = 0; /* number of table spaces */
backupTablespaceList.tablespace = NULL;
backupTargetMediaList.media_type = SQLU_LOCAL_MEDIA;
backupTargetMediaList.sessions = 1; /* number of elements in the target */
backupTargetMediaList.target.media = backupTargetMediaEntries;
strcpy(backupTargetMediaEntries[0].media_entry, serverWorkingPath);
backupParallelismDegree = 1;

backupCallerAction = SQLUB_BACKUP;

/* The API sqlubkp creates a backup copy of a database.
   This API automatically establishes a connection to the specified
    database.
   (This API can also be used to create a backup copy of a table space). */
sqlubkp(dbAlias,
        backupBufferSize,
        backupMode,
        backupType,
        backupCallerAction,
        backupAppId,
        backupStartTimestamp,
        backupBuffersNb,
        &backupTablespaceList,
        &backupTargetMediaList,
        user,
        pswd,
        NULL,
        0,
        NULL,
        backupParallelismDegree,
        &backupImageSize,
        NULL,
        NULL,
        &sqlca);
DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */

  /* depending on the sqlca.sqlcode value, user action may be
     required, such as mounting a new tape */
```

```
    printf("\n  Continuing the backup operation...\n");

    backupCallerAction = SQLUB_CONTINUE;

    /* back up the database */
    sqlubkp(dbAlias,
            backupBufferSize,
            backupMode,
            backupType,
            backupCallerAction,
            backupAppId,
            backupStartTimestamp,
            backupBuffersNb,
            &backupTablespaceList,
            &backupTargetMediaList,
            user,
            pswd,
            NULL,
            0,
            NULL,
            backupParallelismDegree,
            &backupImageSize,
            NULL,
            NULL,
            &sqlca);
    DB2_API_CHECK("Database -- Backup");
  }

  printf("  Backup finished.\n");
  printf("    - backup image size      : %d MB\n", backupImageSize);
  printf("    - backup image path      : %s\n",
         backupTargetMediaEntries[0].media_entry);
  printf("    - backup image time stamp: %s\n",
         backupStartTimestamp);

  /* To restore a remote database, you will first need to create an empty
     database if the client's code page is different from the server's
     code page.
     If this is the case, uncomment the call to DbCreate(). It will create
     an empty database on the server with the server's code page. */

/*
  rc = DbCreate(dbAlias, restoredDbAlias);
  if (rc != 0)
  {
  return rc;
  }
*/

  /*****************************/
  /*    RESTORE THE DATABASE    */
  /*****************************/

  restoreBufferSize = 1024; /*  1024 x 4KB */
  rollforwardPendingState = SQLUD_NOROLLFWD;
```

```
  datalinkMode = SQLUD_NODATALINK;
  restoreType = SQLUD_FULL;
  restoreMode = SQLUD_OFFLINE;
  strcpy(restoreTimestamp, backupStartTimestamp);
  restoreTargetPath = NULL;
  restoreBuffersNb = 1;
  restoreTablespaceList.num_entry = 0; /* number of table spaces */
  restoreTablespaceList.tablespace = NULL;
  restoreSourceMediaList.media_type = SQLU_LOCAL_MEDIA;
  restoreSourceMediaList.sessions = 1; /* number of elements in the target */
  restoreSourceMediaList.target.media = restoreSourceMediaEntries;
  strcpy(restoreSourceMediaEntries[0].media_entry, serverWorkingPath);
  restoreParallelismDegree = 1;

  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);

  restoreCallerAction = SQLUD_RESTORE_STORDEF;

  /* The API sqlurestore is used to restore a database that has been backed
     up using the API sqlubkp. */
  sqlurestore(dbAlias,
              restoredDbAlias,
              restoreBufferSize,
              rollforwardPendingState,
              datalinkMode,
              restoreType,
              restoreMode,
              restoreCallerAction,
              restoreAppId,
              restoreTimestamp,
              restoreTargetPath,
              restoreBuffersNb,
              NULL,
              &restoreTablespaceList,
              &restoreSourceMediaList,
              user,
              pswd,
              NULL,
              0,
              NULL,
              restoreParallelismDegree,
              NULL,
              NULL,
              NULL,
              &sqlca);
/*  DB2_API_CHECK("database restore -- start"); */
  EXPECTED_WARN_CHECK("database restore -- start");

  while (sqlca.sqlcode != 0)
  {
    /* continue the restore operation */
    printf("\n  Continuing the restore operation...\n");
```

```
    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */

    if (sqlca.sqlcode == SQLUD_INACCESSABLE_CONTAINER)
    {
      /* redefine the table space container layout */
      printf("\n  Find and redefine inaccessable containers.\n");
      rc = InaccessableContainersRedefine(serverWorkingPath);
      if (rc != 0)
      {
        return rc;
      }
    }

    restoreCallerAction = SQLUD_CONTINUE;

    /* restore the database */
    sqlurestore(dbAlias,
                restoredDbAlias,
                restoreBufferSize,
                rollforwardPendingState,
                datalinkMode,
                restoreType,
                restoreMode,
                restoreCallerAction,
                restoreAppId,
                restoreTimestamp,
                restoreTargetPath,
                restoreBuffersNb,
                NULL,
                &restoreTablespaceList,
                &restoreSourceMediaList,
                user,
                pswd,
                NULL,
                0,
                NULL,
                restoreParallelismDegree,
                NULL,
                NULL,
                NULL,
                &sqlca);
    DB2_API_CHECK("database restore -- continue");
  }

  printf("\n  Restore finished.\n");

  /* drop the restored database */
  rc = DbDrop(restoredDbAlias);

  return 0;
} /* DbBackupAndRedirectedRestore */

int InaccessableContainersRedefine(char serverWorkingPath[])
```

```
{
  int rc = 0;
  struct sqlca sqlca;
  sqluint32 numTablespaces;
  struct SQLB_TBSPQRY_DATA **ppTablespaces;
  sqluint32 numContainers;
  struct SQLB_TBSCONTQRY_DATA *pContainers;
  int tspNb;
  int contNb;
  char pathSep[2];

  /* The API sqlbmtsq provides a one-call interface to the table space query
     data. The query data for all table spaces in the database is returned
     in an array. */
  sqlbmtsq(&sqlca,
           &numTablespaces,
           &ppTablespaces,
           SQLB_RESERVED1,
           SQLB_RESERVED2);
  DB2_API_CHECK("tablespaces -- get");

  /* refedine the inaccessible 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 inaccessible container */
        printf("\n    Redefine inaccessible 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;
} /* InaccessibleContainersRedefine */

int DbBackupRestoreAndRollforward(char dbAlias[],
                                  char rolledForwardDbAlias[],
                                  char user[],
                                  char pswd[],
                                  char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqlfupd dbCfgParameters[1];
  unsigned short logretain;
  sqluint32 backupBufferSize;
  sqluint32 backupMode;
  sqluint32 backupType;
  sqluint32 backupCallerAction;
  char backupAppId[SQLU_APPLID_LEN + 1];
  char backupStartTimestamp[SQLU_TIME_STAMP_LEN + 1];
  sqluint32 backupBuffersNb;
```

```
struct sqlu_tablespace_bkrst_list backupTablespaceList;
struct sqlu_media_list backupTargetMediaList;
struct sqlu_media_entry backupTargetMediaEntries[1];
sqluint32 backupParallelismDegree;
sqluint32 backupImageSize;
sqluint32 restoreBufferSize;
sqluint32 rollforwardPendingState;
sqluint32 datalinkMode;
sqluint32 restoreType;
sqluint32 restoreMode;
sqluint32 restoreCallerAction;
char restoreAppId[SQLU_APPLID_LEN + 1];
char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];
char *restoreTargetPath;
sqluint32 restoreBuffersNb;
struct sqlu_tablespace_bkrst_list restoreTablespaceList;
struct sqlu_media_list restoreSourceMediaList;
struct sqlu_media_entry restoreSourceMediaEntries[1];
sqluint32 restoreParallelismDegree;
struct rfwd_input rfwdInput;
struct rfwd_output rfwdOutput;
char rollforwardAppId[SQLU_APPLID_LEN + 1];
sqlint32 numReplies;
struct sqlurf_info nodeInfo;

printf("\n**************************\n");
printf("*** ROLLFORWARD RECOVERY ***\n");
printf("**************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  sqlfudb -- Update Database Configuration\n");
printf("  sqlubkp -- Backup Database\n");
printf("  sqlecrea -- Create Database\n");
printf("  sqlurestore -- Restore Database\n");
printf("  sqluroll -- Rollforward Database\n");
printf("  sqledrpd -- Drop Database\n");
printf("TO BACK UP, RESTORE, AND ROLL A DATABASE FORWARD. \n");

printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("    - Enable the configuration parameter LOGRETAIN \n");
printf("        i.e., set LOGRETAIN = RECOVERY/YES\n");


dbCfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
dbCfgParameters[0].ptrvalue = (char *)&logretain;

/* enable the configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_RECOVERY;

/* The API sqlfudb is used to modify individual entries in a specific
   database configuration file. */
sqlfudb(dbAlias, 1, dbCfgParameters, &sqlca);
DB2_API_CHECK("Db Log Retain -- Enable");

/* start the backup operation */
printf("\n  Backing up the '%s' database...\n", dbAlias);
```

```
backupBufferSize = 16; /*  16 x 4KB */
backupMode = SQLUB_OFFLINE;
backupType = SQLUB_FULL;
backupBuffersNb = 1;
backupTablespaceList.num_entry = 0; /* number of table spaces */
backupTablespaceList.tablespace = NULL;
backupTargetMediaList.media_type = SQLU_LOCAL_MEDIA;
backupTargetMediaList.sessions = 1; /* number of elements in the target */
backupTargetMediaList.target.media = backupTargetMediaEntries;
strcpy(backupTargetMediaEntries[0].media_entry, serverWorkingPath);
backupParallelismDegree = 1;

backupCallerAction = SQLUB_BACKUP;

/* The API sqlubkp creates a backup copy of a database.
   This API automatically establishes a connection to the specified
    database.
   (This API can also be used to create a backup copy of a table space). */
sqlubkp(dbAlias,
        backupBufferSize,
        backupMode,
        backupType,
        backupCallerAction,
        backupAppId,
        backupStartTimestamp,
        backupBuffersNb,
        &backupTablespaceList,
        &backupTargetMediaList,
        user,
        pswd,
        NULL,
        0,
        NULL,
        backupParallelismDegree,
        &backupImageSize,
        NULL,
        NULL,
        &sqlca);
DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */
  printf("\n  Continuing the backup operation...\n");

  /* depending on the sqlca.sqlcode value, user action may be
     required, such as mounting a new tape. */

  backupCallerAction = SQLUB_CONTINUE;

  /* back up the database */
  sqlubkp(dbAlias,
          backupBufferSize,
          backupMode,
```

```
                backupType,
                backupCallerAction,
                backupAppId,
                backupStartTimestamp,
                backupBuffersNb,
                &backupTablespaceList,
                &backupTargetMediaList,
                user,
                pswd,
                NULL,
                0,
                NULL,
                backupParallelismDegree,
                &backupImageSize,
                NULL,
                NULL,
                &sqlca);
    DB2_API_CHECK("Database -- Backup");
  }

  printf("  Backup finished.\n");
  printf("    - backup image size      : %d MB\n", backupImageSize);
  printf("    - backup image path      : %s\n",
         backupTargetMediaEntries[0].media_entry);
  printf("    - backup image time stamp: %s\n",
         backupStartTimestamp);

  /* To restore a remote database, you will first need to create an empty
     database if the client's code page is different from the server's
     code page.
     If this is the case, uncomment the call to DbCreate(). It will create
     an empty database on the server with the server's code page. */

/*
  rc = DbCreate(dbAlias, rolledForwardDbAlias);
  if (rc != 0)
  {
    return rc;
  }
*/

  /*****************************/
  /*    RESTORE THE DATABASE    */
  /*****************************/

  restoreBufferSize = 1024; /*  1024 x 4KB */
  rollforwardPendingState = SQLUD_ROLLFWD;
  datalinkMode = SQLUD_NODATALINK;
  restoreType = SQLUD_FULL;
  restoreMode = SQLUD_OFFLINE;
  strcpy(restoreTimestamp, backupStartTimestamp);
  restoreTargetPath = NULL;
  restoreBuffersNb = 1;
  restoreTablespaceList.num_entry = 0; /* number of table spaces */
  restoreTablespaceList.tablespace = NULL;
```

```
restoreSourceMediaList.media_type = SQLU_LOCAL_MEDIA;
restoreSourceMediaList.sessions = 1; /* number of elements in the target */
restoreSourceMediaList.target.media = restoreSourceMediaEntries;
strcpy(restoreSourceMediaEntries[0].media_entry, serverWorkingPath);
restoreParallelismDegree = 1;

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);

restoreCallerAction = SQLUD_RESTORE;

/* The API sqlurestore is used to restore a database that has been backed
   up using the API sqlubkp. */
sqlurestore(dbAlias,
            rolledForwardDbAlias,
            restoreBufferSize,
            rollforwardPendingState,
            datalinkMode,
            restoreType,
            restoreMode,
            restoreCallerAction,
            restoreAppId,
            restoreTimestamp,
            restoreTargetPath,
            restoreBuffersNb,
            NULL,
            &restoreTablespaceList,
            &restoreSourceMediaList,
            user,
            pswd,
            NULL,
            0,
            NULL,
            restoreParallelismDegree,
            NULL,
            NULL,
            NULL,
            &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. */

  restoreCallerAction = SQLUD_CONTINUE;

  /* restore the database */
  sqlurestore(dbAlias,
              rolledForwardDbAlias,
```

```
                restoreBufferSize,
                rollforwardPendingState,
                datalinkMode,
                restoreType,
                restoreMode,
                restoreCallerAction,
                restoreAppId,
                restoreTimestamp,
                restoreTargetPath,
                restoreBuffersNb,
                NULL,
                &restoreTablespaceList,
                &restoreSourceMediaList,
                user,
                pswd,
                NULL,
                0,
                NULL,
                restoreParallelismDegree,
                NULL,
                NULL,
                NULL,
                &sqlca);
  DB2_API_CHECK("database restore -- continue");
}

printf("\n  Restore finished.\n");


/*****************************/
/*    ROLLFORWARD RECOVERY    */
/*****************************/

printf("\n  Rolling '%s' database forward ...\n", rolledForwardDbAlias);

rfwdInput.version = SQLUM_RFWD_VERSION;
rfwdInput.pDbAlias = rolledForwardDbAlias;
rfwdInput.CallerAction = SQLUM_ROLLFWD_STOP;
rfwdInput.pStopTime = SQLUM_INFINITY_TIMESTAMP;
rfwdInput.pUserName = user;
rfwdInput.pPassword = pswd;
rfwdInput.pOverflowLogPath = serverWorkingPath;
rfwdInput.NumChngLgOvrflw = 0;
rfwdInput.pChngLogOvrflw = NULL;
rfwdInput.ConnectMode = SQLUM_OFFLINE;
rfwdInput.pTablespaceList = NULL;
rfwdInput.AllNodeFlag = SQLURF_ALL_NODES;
rfwdInput.NumNodes = 0;
rfwdInput.pNodeList = NULL;
rfwdInput.NumNodeInfo = 1;
rfwdInput.DlMode = 0;
rfwdInput.pReportFile = NULL;
rfwdInput.pDroppedTblID = NULL;
rfwdInput.pExportDir = NULL;
```

```
  rfwdOutput.pApplicationId = rollforwardAppId;
  rfwdOutput.pNumReplies = &numReplies;
  rfwdOutput.pNodeInfo = &nodeInfo;

  /* rollforward database */
  /* The API sqluroll rollforward recovers a database by
     applying transactions recorded in the database log files. */
  sqluroll(&rfwdInput, &rfwdOutput, &sqlca);
  DB2_API_CHECK("rollforward -- start");

  printf("  Rollforward finished.\n");

  /* drop the restored database */
  rc = DbDrop(rolledForwardDbAlias);

  return 0;
} /* DbBackupRestoreAndRollforward */

int DbLogRecordsForCurrentConnectionRead(char dbAlias[],
                                         char user[],
                                         char pswd[],
                                         char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqlfupd dbCfgParameters[1];
  unsigned short logretain;
  sqluint32 backupBufferSize;
  sqluint32 backupMode;
  sqluint32 backupType;
  sqluint32 backupCallerAction;
  char backupAppId[SQLU_APPLID_LEN + 1];
  char backupStartTimestamp[SQLU_TIME_STAMP_LEN + 1];
  sqluint32 backupBuffersNb;
  struct sqlu_tablespace_bkrst_list backupTablespaceList;
  struct sqlu_media_list backupTargetMediaList;
  struct sqlu_media_entry backupTargetMediaEntries[1];
  sqluint32 backupParallelismDegree;
  sqluint32 backupImageSize;
  SQLU_LSN startLSN;
  SQLU_LSN endLSN;
  char *logBuffer;
  sqluint32 logBufferSize;
  SQLU_RLOG_INFO readLogInfo;
  int i;

  printf("\n*****************************\n");
  printf("*** ASYNCHRONOUS READ LOG ***\n");
  printf("*****************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  sqlfudb -- Update Database Configuration\n");
  printf("  sqlubkp -- Backup Database\n");
  printf("  sqlurlog -- 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");

dbCfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
dbCfgParameters[0].ptrvalue = (char *)&logretain;

/* enable LOGRETAIN */
logretain = SQLF_LOGRETAIN_RECOVERY;

/* The API sqlfudb is used to modify individual entries in a specific
   database configuration file. */
sqlfudb(dbAlias, 1, dbCfgParameters, &sqlca);
DB2_API_CHECK("Db Log Retain -- Enable");

/* start the backup operation */
printf("\n  Backing up the '%s' database...\n", dbAlias);

backupBufferSize = 16; /*  16 x 4KB */
backupMode = SQLUB_OFFLINE;
backupType = SQLUB_FULL;
backupBuffersNb = 1;
backupTablespaceList.num_entry = 0; /* number of table spaces */
backupTablespaceList.tablespace = NULL;
backupTargetMediaList.media_type = SQLU_LOCAL_MEDIA;
backupTargetMediaList.sessions = 1; /* number of elements in the target */
backupTargetMediaList.target.media = backupTargetMediaEntries;
strcpy(backupTargetMediaEntries[0].media_entry, serverWorkingPath);
backupParallelismDegree = 1;

backupCallerAction = SQLUB_BACKUP;

/* The API sqlubkp creates a backup copy of a database.
   This API automatically establishes a connection to the specified
    database.
   (This API can also be used to create a backup copy of a table space). */
sqlubkp(dbAlias,
        backupBufferSize,
        backupMode,
        backupType,
        backupCallerAction,
        backupAppId,
        backupStartTimestamp,
        backupBuffersNb,
        &backupTablespaceList,
        &backupTargetMediaList,
        user,
```

```
        pswd,
        NULL,
        0,
        NULL,
        backupParallelismDegree,
        &backupImageSize,
        NULL,
        NULL,
        &sqlca);
DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */
  printf("\n  Continuing the backup operation...\n");

  /* Depending on the sqlca.sqlcode value, user action may be
     required, such as mounting a new tape. */

  backupCallerAction = SQLUB_CONTINUE;

  /* back up the database */
  sqlubkp(dbAlias,
          backupBufferSize,
          backupMode,
          backupType,
          backupCallerAction,
          backupAppId,
          backupStartTimestamp,
          backupBuffersNb,
          &backupTablespaceList,
          &backupTargetMediaList,
          user,
          pswd,
          NULL,
          0,
          NULL,
          backupParallelismDegree,
          &backupImageSize,
          NULL,
          NULL,
          &sqlca);
  DB2_API_CHECK("Database -- Backup");
}

printf("  Backup finished.\n");
printf("    - backup image size      : %d MB\n", backupImageSize);
printf("    - backup image path      : %s\n",
       backupTargetMediaEntries[0].media_entry);
printf("    - backup image time stamp: %s\n",
       backupStartTimestamp);

/* connect to the database */
rc = DbConn(dbAlias, user, pswd);
if (rc != 0)
```

```
{
  return rc;
}

/* invoke SQL statements to fill database log */
printf("\n  Invoke the following SQL statements:\n"
       "    ALTER TABLE emp_resume DATA CAPTURE CHANGES;\n"
       "    COMMIT;\n"
       "    INSERT INTO emp_resume\n"
       "      VALUES('000777', 'ascii', 'This is a new resume.');\n"
       "            ('777777', 'ascii', 'This is another new resume');\n"
       "    COMMIT;\n"
       "    DELETE FROM emp_resume WHERE empno = '000777';\n"
       "    DELETE FROM emp_resume WHERE empno = '777777';\n"
       "    COMMIT;\n"
       "    DELETE FROM emp_resume WHERE empno = '000140';\n"
       "    ROLLBACK;\n"
       "    ALTER TABLE emp_resume DATA CAPTURE NONE;\n"
       "    COMMIT;\n");

EXEC SQL ALTER TABLE emp_resume DATA CAPTURE CHANGES;
EMB_SQL_CHECK("SQL statement 1 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 2 -- invoke");

EXEC SQL INSERT INTO emp_resume
  VALUES('000777', 'ascii', 'This is a new resume.'),
        ('777777', 'ascii', 'This is another new resume');
EMB_SQL_CHECK("SQL statement 3 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 4 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '000777';
EMB_SQL_CHECK("SQL statement 5 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '777777';
EMB_SQL_CHECK("SQL statement 6 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 7 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '000140';
EMB_SQL_CHECK("SQL statement 8 -- invoke");

EXEC SQL ROLLBACK;
EMB_SQL_CHECK("SQL statement 9 -- invoke");

EXEC SQL ALTER TABLE emp_resume DATA CAPTURE NONE;
EMB_SQL_CHECK("SQL statement 10 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 11 -- invoke");
```

```
  printf("\n  Start reading database log.\n");

  logBuffer = NULL;
  logBufferSize = 0;

  /* The API sqlurlog (Asynchronours 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: */
  rc = sqlurlog(SQLU_RLOG_QUERY,
                &startLSN,
                &endLSN,
                logBuffer,
                logBufferSize,
                &readLogInfo,
                &sqlca);
/*  DB2_API_CHECK("database log info -- get"); */
  EXPECTED_WARN_CHECK("database log info -- get");

  logBufferSize = 64 * 1024;
  logBuffer = (char *)malloc(logBufferSize);

  memcpy(&startLSN, &(readLogInfo.initialLSN), sizeof(startLSN));
  memcpy(&endLSN, &(readLogInfo.curActiveLSN), sizeof(endLSN));

  /* Extract a log record from the database logs, and
     read the first log sequence asynchronously: */
  rc = sqlurlog(SQLU_RLOG_READ,
                &startLSN,
                &endLSN,
                logBuffer,
                logBufferSize,
                &readLogInfo,
                &sqlca);
  if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
  {
    DB2_API_CHECK("database logs -- read");
  }
  else
  {
    if (readLogInfo.logRecsWritten == 0)
    {
      printf("\n  Database log empty.\n");
    }
  }

  /* display log buffer */
  rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);

  while (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
  {
    /* read the next log sequence */
```

```
    memcpy(&startLSN, &(readLogInfo.lastReadLSN), sizeof(startLSN));
    /* increase startLSN by 1 */
    startLSN.lsnChar[5] = startLSN.lsnChar[5] + 1;
    i = 5;
    while (startLSN.lsnChar[i] == 0 && i > 0)
    {
      startLSN.lsnChar[i - 1] = startLSN.lsnChar[i - 1] + 1;
      i = i - 1;
    }

    /* Extract a log record from the database logs, and
     read the next log sequence asynchronously: */
    rc = sqlurlog(SQLU_RLOG_READ,
                  &startLSN,
                  &endLSN,
                  logBuffer,
                  logBufferSize,
                  &readLogInfo,
                  &sqlca);
    if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
    {
      DB2_API_CHECK("database logs -- read");
    }

    /* display log buffer */
    rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);
  }

  /* free the log buffer */
  free(logBuffer);

  /* disconnect from the database */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* DbLogRecordsForCurrentConnectionRead */

int LogBufferDisplay(char *logBuffer, sqluint32 numLogRecords)
{
  int rc = 0;
  sqluint32 logRecordNb;
  sqluint32 recordSize;
  sqluint16 recordType;
  sqluint16 recordFlag;
  char *recordBuffer;

  /* initialize recordBuffer */
  recordBuffer = logBuffer + sizeof(SQLU_LSN);

  for (logRecordNb = 0; logRecordNb < numLogRecords; logRecordNb++)
  {
```

```
    recordSize = *(sqluint32 *)(recordBuffer);
    recordType = *(sqluint16 *)(recordBuffer + 4);
    recordFlag = *(sqluint16 *)(recordBuffer + 6);

    rc = LogRecordDisplay(recordBuffer, recordSize, recordType, recordFlag);
    /* update recordBuffer */
    recordBuffer = recordBuffer + recordSize + sizeof(SQLU_LSN);
  }

  return 0;
} /* LogBufferDisplay */

int LogRecordDisplay(char *recordBuffer,
                     sqluint32 recordSize,
                     sqluint16 recordType,
                     sqluint16 recordFlag)
{
  int rc = 0;
  sqluint32 logManagerLogRecordHeaderSize;
  char *recordDataBuffer;
  sqluint32 recordDataSize;
  char *recordHeaderBuffer;
  sqluint8 componentIdentifier;
  sqluint32 recordHeaderSize;

  /* determine logManagerLogRecordHeaderSize */
  if ((char)recordType == 'C')
  { /* compensation */
    if (recordFlag & 0x0002)
    { /* propagatable */
      logManagerLogRecordHeaderSize = 32;
    }
    else
    {
      logManagerLogRecordHeaderSize = 26;
    }
  }
  else
  { /* non compensation */
    logManagerLogRecordHeaderSize = 20;
  }

  switch ((char)recordType)
  {
    case 'E':
    case 'M':
    case 'A':
      recordDataBuffer = recordBuffer + logManagerLogRecordHeaderSize;
      recordDataSize = recordSize - logManagerLogRecordHeaderSize;
      rc = SimpleLogRecordDisplay(recordType,
                                  recordFlag,
                                  recordDataBuffer,
                                  recordDataSize);
      break;
    case 'N':
```

```
    case 'C':
      recordHeaderBuffer = recordBuffer + logManagerLogRecordHeaderSize;
      componentIdentifier = *(sqluint8 *)recordHeaderBuffer;
      switch (componentIdentifier)
      {
        case 1:
          recordHeaderSize = 6;
          break;
        default:
          printf("    Unknown complex log record: %lu %c %u\n",
                 recordSize, recordType, componentIdentifier);
          return 1;
      }
      recordDataBuffer = recordBuffer +
                         logManagerLogRecordHeaderSize +
                         recordHeaderSize;
      recordDataSize = recordSize -
                       logManagerLogRecordHeaderSize -
                       recordHeaderSize;
      rc = ComplexLogRecordDisplay(recordType,
                                   recordFlag,
                                   recordHeaderBuffer,
                                   recordHeaderSize,
                                   componentIdentifier,
                                   recordDataBuffer,
                                   recordDataSize);
      break;
    default:
      printf("    Unknown log record: %lu %c\n",
             recordSize, (char)recordType);
      break;
  }

  return 0;
} /* LogRecordDisplay */

int SimpleLogRecordDisplay(sqluint16 recordType,
                           sqluint16 recordFlag,
                           char *recordDataBuffer,
                           sqluint32 recordDataSize)
{
  int rc = 0;
  sqluint32 timeTransactionCommited;
  sqluint16 authIdLen;
  char authId[129];

  switch ((char)recordType)
  {
    case 'E':
      printf("\n    Record type: Local pending list\n");
      timeTransactionCommited = *(sqluint32 *)(recordDataBuffer);
      authIdLen = *(sqluint16 *)(recordDataBuffer + 4);
      memcpy(authId, (char *)(recordDataBuffer + 6), authIdLen);
      authId[authIdLen] = '\0';
      printf("      %s: %lu\n",
```

```
              "UTC transaction committed (in seconds since 01/01/70)",
              timeTransactionCommited);
        printf("      authorization ID of the application: %s\n", authId);
        break;
      case 'M':
        printf("\n    Record type: Normal commit\n");
        timeTransactionCommited = *(sqluint32 *)(recordDataBuffer);
        authIdLen = (sqluint16) (recordDataSize - 4);
        memcpy(authId, (char *)(recordDataBuffer + 4), authIdLen);
        authId[authIdLen] = '\0';
        printf("      %s: %lu\n",
              "UTC transaction committed (in seconds since 01/01/70)",
              timeTransactionCommited);
        printf("      authorization ID of the application: %s\n", authId);
        break;
      case 'A':
        printf("\n    Record type: Normal abort\n");
        authIdLen = (sqluint16) (recordDataSize);
        memcpy(authId, (char *)(recordDataBuffer), authIdLen);
        authId[authIdLen] = '\0';
        printf("      authorization ID of the application: %s\n", authId);
        break;
      default:
        printf("    Unknown simple log record: %c %lu\n",
              (char)recordType, recordDataSize);
        break;
    }

    return 0;
} /* SimpleLogRecordDisplay */

int ComplexLogRecordDisplay(sqluint16 recordType,
                            sqluint16 recordFlag,
                            char *recordHeaderBuffer,
                            sqluint32 recordHeaderSize,
                            sqluint8 componentIdentifier,
                            char *recordDataBuffer,
                            sqluint32 recordDataSize)
{
  int rc = 0;
  sqluint8 functionIdentifier;
  /* for insert, delete, undo delete */
  sqluint32 RID;
  sqluint16 subRecordLen;
  sqluint16 subRecordOffset;
  char *subRecordBuffer;
  /* for update */
  sqluint32 newRID;
  sqluint16 newSubRecordLen;
  sqluint16 newSubRecordOffset;
  char *newSubRecordBuffer;
  sqluint32 oldRID;
  sqluint16 oldSubRecordLen;
  sqluint16 oldSubRecordOffset;
  char *oldSubRecordBuffer;
```

```
/* for alter table attributes */
sqluint32 alterBitMask;
sqluint32 alterBitValues;

switch ((char)recordType)
{
  case 'N':
    printf("\n    Record type: Normal\n");
    break;
  case 'C':
    printf("\n    Record type: Compensation\n");
    break;
  default:
    printf("\n    Unknown complex log record type: %c\n", recordType);
    break;
}

switch (componentIdentifier)
{
  case 1:
    printf("      component ID: DMS log record\n");
    break;
  default:
    printf("      unknown component ID: %d\n", componentIdentifier);
    break;
}

functionIdentifier = *(sqluint8 *)(recordHeaderBuffer + 1);
switch (functionIdentifier)
{
  case 106:
    printf("      function ID: Delete Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 111:
    printf("      function ID: Undo Delete Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 118:
    printf("      function ID: Insert Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
```

```
  subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
  subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
  printf("        RID: %lu\n", RID);
  printf("        subrecord length: %u\n", subRecordLen);
  printf("        subrecord offset: %u\n", subRecordOffset);
  subRecordBuffer = recordDataBuffer + 12;
  rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
  break;
case 120:
  printf("      function ID: Update Record\n");
  oldRID = *(sqluint32 *)(recordDataBuffer + 2);
  oldSubRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
  oldSubRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
  newRID = *(sqluint32 *)(recordDataBuffer +
                          12 + oldSubRecordLen + recordHeaderSize + 2);
  newSubRecordLen = *(sqluint16 *)(recordDataBuffer +
                                   12 + oldSubRecordLen +
                                   recordHeaderSize + 6);
  newSubRecordOffset =
    *(sqluint16 *)(recordDataBuffer + 12 + oldSubRecordLen +
                   recordHeaderSize + 10);
  printf("        oldRID: %lu\n", oldRID);
  printf("        old subrecord length: %u\n", oldSubRecordLen);
  printf("        old subrecord offset: %u\n",
         oldSubRecordOffset);
  oldSubRecordBuffer = recordDataBuffer + 12;
  rc = LogSubRecordDisplay(oldSubRecordBuffer, oldSubRecordLen);
  printf("        newRID: %lu\n", newRID);
  printf("        new subrecord length: %u\n", newSubRecordLen);
  printf("        new subrecord offset: %u\n",
         newSubRecordOffset);
  newSubRecordBuffer = recordDataBuffer +
    12 + oldSubRecordLen + recordHeaderSize + 12;
  rc = LogSubRecordDisplay(newSubRecordBuffer, newSubRecordLen);
  break;
case 124:
  printf("      function ID:  Alter Table Attribute\n");
  alterBitMask = *(sqluint32 *)(recordDataBuffer + 2);
  alterBitValues = *(sqluint32 *)(recordDataBuffer + 6);
  if (alterBitMask & 0x00000001)
  {
    /* Alter the value of the 'propagation' attribute: */
    printf("        Propagation attribute is changed to: ");
    if (alterBitValues & 0x00000001)
    {
      printf("ON\n");
    }
    else
    {
      printf("OFF\n");
    }
  }
  if (alterBitMask & 0x00000002)
  {
    /* Alter the value of the 'pending' attribute: */
```

```
      printf("        Pending attribute is changed to: ");
      if (alterBitValues & 0x00000002)
      {
        printf("ON\n");
      }
      else
      {
        printf("OFF\n");
      }
    }
    if (alterBitMask & 0x00010000)
    {
      /* Alter the value of the 'append mode' attribute: */
      printf("        Append Mode attribute is changed to: ");
      if (alterBitValues & 0x00010000)
      {
        printf("ON\n");
      }
      else
      {
        printf("OFF\n");
      }
    }
    if (alterBitMask & 0x00200000)
    {
      /* Alter the value of the 'LF Propagation' attribute: */
      printf("        LF Propagation attribute is changed to: ");
      if (alterBitValues & 0x00200000)
      {
        printf("ON\n");
      }
      else
      {
        printf("OFF\n");
      }
    }
    if (alterBitMask & 0x00400000)
    {
      /* Alter the value of the 'LOB Propagation' attribute: */
      printf("        LOB Propagation attribute is changed to: ");
      if (alterBitValues & 0x00400000)
      {
        printf("ON\n");
      }
      else
      {
        printf("OFF\n");
      }
    }
    break;
  default:
    printf("      unknown function identifier: %u\n",
           functionIdentifier);
    break;
}
```

```
  return 0;
} /* ComplexLogRecordDisplay */

int LogSubRecordDisplay(char *recordBuffer, sqluint16 recordSize)
{
  int rc = 0;
  sqluint8 recordType;
  sqluint8 updatableRecordType;
  sqluint16 userDataFixedLength;
  char *userDataBuffer;
  sqluint16 userDataSize;

  recordType = *(sqluint8 *)(recordBuffer);
  if (recordType != 0 && recordType != 4)
  {
    printf("        Unknown subrecord type.\n");
  }
  else if (recordType == 4)
  {
    printf("        subrecord type: Special control\n");
  }
  else
  {
    /* recordType == 0 */
    printf("        subrecord type: Updatable, ");
    updatableRecordType = *(sqluint8 *)(recordBuffer + 4);
    if (updatableRecordType != 1)
    {
      printf("Internal control\n");
    }
    else
    {
      printf("Formatted user data\n");
      userDataFixedLength = *(sqluint16 *)(recordBuffer + 6);
      printf("        user data fixed length: %u\n",
             userDataFixedLength);
      userDataBuffer = recordBuffer + 8;
      userDataSize = recordSize - 8;
      rc = UserDataDisplay(userDataBuffer, userDataSize);
    }
  }

  return 0;
} /* LogSubRecordDisplay */

int UserDataDisplay(char *dataBuffer, sqluint16 dataSize)
{
  int rc = 0;

  sqluint16 line, col;

  printf("        user data:\n");

  for (line = 0; line * 10 < dataSize; line = line + 1)
```

```
  {
    printf("        ");
    for (col = 0; col < 10; col = col + 1)
    {
      if (line * 10 + col < dataSize)
      {
        printf("%02X ", dataBuffer[line * 10 + col]);
      }
      else
      {
        printf("   ");
      }
    }
    printf("*");
    for (col = 0; col < 10; col = col + 1)
    {
      if (line * 10 + col < dataSize)
      {
        if (isalpha(dataBuffer[line * 10 + col]) ||
            isdigit(dataBuffer[line * 10 + col]))
        {
          printf("%c", dataBuffer[line * 10 + col]);
        }
        else
        {
          printf(".");
        }
      }
      else
      {
        printf(" ");
      }
    }
    printf("*");
    printf("\n");
  }

  return 0;
} /* UserDataDisplay */

int DbRecoveryHistoryFileRead(char dbAlias[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint32 numEntries;
  sqluint16 recoveryHistoryFileHandle;
  sqluint32 entryNb;
  struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
  struct db2HistoryData histEntryData;

  printf("\n*********************************************\n");
  printf("*** READ A DATABASE RECOVERY HISTORY FILE ***\n");
  printf("*********************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
```

```
printf("  db2HistoryOpenScan -- Open Recovery History File Scan\n");
printf("  db2HistoryGetEntry -- Get Next Recovery History File Entry\n");
printf("  db2HistoryCloseScan -- Close Recovery History File Scan\n");
printf("TO READ A DATABASE RECOVERY HISTORY FILE.\n");

/* initialize the data structures */
dbHistoryOpenParam.piDatabaseAlias = dbAlias;
dbHistoryOpenParam.piTimestamp = NULL;
dbHistoryOpenParam.piObjectName = NULL;
dbHistoryOpenParam.iCallerAction = DB2HISTORY_LIST_HISTORY;

dbHistoryEntryGetParam.pioHistData = &histEntryData;
dbHistoryEntryGetParam.iCallerAction = DB2HISTORY_GET_ALL;
rc = HistoryEntryDataFieldsAlloc(&histEntryData);
if (rc != 0)
{
  return rc;
}

/*****************************************/
/* OPEN THE DATABASE RECOVERY HISTORY FILE */
/*****************************************/
printf("\n  Open recovery history file for '%s' database.\n", dbAlias);

/* open the recovery history file to scan */
db2HistoryOpenScan(db2Version710, &dbHistoryOpenParam, &sqlca);
DB2_API_CHECK("database recovery history file -- open");

numEntries = dbHistoryOpenParam.oNumRows;

/* dbHistoryOpenParam.oHandle returns the handle for scan access */
recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;
dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;

/*********************************************/
/* READ AN ENTRY IN THE RECOVERY HISTORY FILE */
/*********************************************/
for (entryNb = 0; entryNb < numEntries; entryNb = entryNb + 1)
{
  printf("\n  Read entry number %u.\n", entryNb);

  /* get the next entry from the recovery history file */
  db2HistoryGetEntry(db2Version710, &dbHistoryEntryGetParam, &sqlca);
  DB2_API_CHECK("database recovery history file entry -- read")

  /* display the entries in the recovery history file */
  printf("\n  Display entry number %u.\n", entryNb);
  rc = HistoryEntryDisplay(histEntryData);
}

/*****************************************/
/* CLOSE THE DATABASE RECOVERY HISTORY FILE */
/*****************************************/
printf("\n  Close recovery history file for '%s' database.\n", dbAlias);
```

```
  /* The API db2HistoryCloseScan ends the recovery history file scan and
     frees DB2 resources required for the scan. */
  db2HistoryCloseScan(db2Version710, &recoveryHistoryFileHandle, &sqlca);
  DB2_API_CHECK("database recovery history file -- close");

  /* free the allocated memory */
  rc = HistoryEntryDataFieldsFree(&histEntryData);

  return 0;
} /* DbRecoveryHistoryFileRead */

int HistoryEntryDataFieldsAlloc(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb;

  strcpy(pHistEntryData->ioHistDataID, "SQLUHINF");

  pHistEntryData->oObjectPart.pioData = malloc(17 + 1);
  pHistEntryData->oObjectPart.iLength = 17 + 1;

  pHistEntryData->oEndTime.pioData = malloc(12 + 1);
  pHistEntryData->oEndTime.iLength = 12 + 1;

  pHistEntryData->oFirstLog.pioData = malloc(8 + 1);
  pHistEntryData->oFirstLog.iLength = 8 + 1;

  pHistEntryData->oLastLog.pioData = malloc(8 + 1);
  pHistEntryData->oLastLog.iLength = 8 + 1;

  pHistEntryData->oID.pioData = malloc(128 + 1);
  pHistEntryData->oID.iLength = 128 + 1;

  pHistEntryData->oTableQualifier.pioData = malloc(128 + 1);
  pHistEntryData->oTableQualifier.iLength = 128 + 1;

  pHistEntryData->oTableName.pioData = malloc(128 + 1);
  pHistEntryData->oTableName.iLength = 128 + 1;

  pHistEntryData->oLocation.pioData = malloc(128 + 1);
  pHistEntryData->oLocation.iLength = 128 + 1;

  pHistEntryData->oComment.pioData = malloc(128 + 1);
  pHistEntryData->oComment.iLength = 128 + 1;

  pHistEntryData->oCommandText.pioData = malloc(128 + 1);
  pHistEntryData->oCommandText.iLength = 128 + 1;

  pHistEntryData->poEventSQLCA =
    (struct sqlca *)malloc(sizeof(struct sqlca));

  pHistEntryData->poTablespace = (db2Char *)malloc(3 * sizeof(db2Char));
  for (tsNb = 0; tsNb < 3; tsNb = tsNb + 1)
  {
    pHistEntryData->poTablespace[tsNb].pioData = malloc(18 + 1);
```

```
    pHistEntryData->poTablespace[tsNb].iLength = 18 + 1;
  }

  pHistEntryData->iNumTablespaces = 3;

  return 0;
} /* HistoryEntryDataFieldsAlloc */

int HistoryEntryDisplay(struct db2HistoryData histEntryData)
{
  int rc = 0;
  char buf[129];
  sqluint32 tsNb;

  memcpy(buf, histEntryData.oObjectPart.pioData,
              histEntryData.oObjectPart.oLength);
  buf[histEntryData.oObjectPart.oLength] = '\0';
  printf("    object part: %s\n", buf);

  memcpy(buf, histEntryData.oEndTime.pioData,
              histEntryData.oEndTime.oLength);
  buf[histEntryData.oEndTime.oLength] = '\0';
  printf("    end time: %s\n", buf);

  memcpy(buf, histEntryData.oFirstLog.pioData,
              histEntryData.oFirstLog.oLength);
  buf[histEntryData.oFirstLog.oLength] = '\0';
  printf("    first log: %s\n", buf);

  memcpy(buf, histEntryData.oLastLog.pioData,
              histEntryData.oLastLog.oLength);
  buf[histEntryData.oLastLog.oLength] = '\0';
  printf("    last log: %s\n", buf);

  memcpy(buf, histEntryData.oID.pioData, histEntryData.oID.oLength);
  buf[histEntryData.oID.oLength] = '\0';
  printf("    ID: %s\n", buf);

  memcpy(buf, histEntryData.oTableQualifier.pioData,
              histEntryData.oTableQualifier.oLength);
  buf[histEntryData.oTableQualifier.oLength] = '\0';
  printf("    table qualifier: %s\n", buf);

  memcpy(buf, histEntryData.oTableName.pioData,
              histEntryData.oTableName.oLength);
  buf[histEntryData.oTableName.oLength] = '\0';
  printf("    table name: %s\n", buf);

  memcpy(buf, histEntryData.oLocation.pioData,
              histEntryData.oLocation.oLength);
  buf[histEntryData.oLocation.oLength] = '\0';
  printf("    location: %s\n", buf);

  memcpy(buf, histEntryData.oComment.pioData,
              histEntryData.oComment.oLength);
```

```
  buf[histEntryData.oComment.oLength] = '\0';
  printf("    comment: %s\n", buf);

  memcpy(buf, histEntryData.oCommandText.pioData,
              histEntryData.oCommandText.oLength);
  buf[histEntryData.oCommandText.oLength] = '\0';
  printf("    command text: %s\n", buf);
  printf("    history file entry ID: %u\n", histEntryData.oEID.ioHID);
  printf("    table spaces:\n");

  for (tsNb = 0; tsNb < histEntryData.oNumTablespaces; tsNb = tsNb + 1)
  {
    memcpy(buf, histEntryData.poTablespace[tsNb].pioData,
                histEntryData.poTablespace[tsNb].oLength);
    buf[histEntryData.poTablespace[tsNb].oLength] = '\0';
    printf("      %s\n", buf);
  }

  printf("    type of operation: %c\n", histEntryData.oOperation);
  printf("    granularity of the operation: %c\n", histEntryData.oObject);
  printf("    operation type: %c\n", histEntryData.oOptype);
  printf("    entry status: %c\n", histEntryData.oStatus);
  printf("    device type: %c\n", histEntryData.oDeviceType);
  printf("    SQLCA:\n");
  printf("      sqlcode: %ld\n", histEntryData.poEventSQLCA->sqlcode);
  memcpy(buf, histEntryData.poEventSQLCA->sqlstate, 5);
  buf[5] = '\0';
  printf("      sqlstate: %s\n", buf);
  memcpy(buf, histEntryData.poEventSQLCA->sqlerrmc,
              histEntryData.poEventSQLCA->sqlerrml);
  buf[histEntryData.poEventSQLCA->sqlerrml] = '\0';
  printf("      message: %s\n", buf);

  return 0;
} /* HistoryEntryDisplay */

int HistoryEntryDataFieldsFree(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb;

  free(pHistEntryData->oObjectPart.pioData);
  free(pHistEntryData->oEndTime.pioData);
  free(pHistEntryData->oFirstLog.pioData);
  free(pHistEntryData->oLastLog.pioData);
  free(pHistEntryData->oID.pioData);
  free(pHistEntryData->oTableQualifier.pioData);
  free(pHistEntryData->oTableName.pioData);
  free(pHistEntryData->oLocation.pioData);
  free(pHistEntryData->oComment.pioData);
  free(pHistEntryData->oCommandText.pioData);
  free(pHistEntryData->poEventSQLCA);

  for (tsNb = 0; tsNb < 3; tsNb = tsNb + 1)
  {
```

```
    free(pHistEntryData->poTablespace[tsNb].pioData);
  }

  free(pHistEntryData->poTablespace);

  return 0;
} /* HistoryEntryDataFieldsFree */

int DbFirstRecoveryHistoryFileEntryUpdate(char dbAlias[],
                                          char user[],
                                          char pswd[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint16 recoveryHistoryFileHandle;
  struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
  struct db2HistoryData histEntryData;
  char newLocation[DB2HISTORY_LOCATION_SZ + 1];
  char newComment[DB2HISTORY_COMMENT_SZ + 1];
  struct db2HistoryUpdateStruct dbHistoryUpdateParam;

  printf("\n*****************************************************\n");
  printf("*** UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY ***\n");
  printf("*****************************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2HistoryOpenScan -- Open Recovery History File Scan\n");
  printf("  db2HistoryGetEntry -- Get Next Recovery History File Entry\n");
  printf("  db2HistoryUpdate -- Update Recovery History File\n");
  printf("  db2HistoryCloseScan -- Close Recovery History File Scan\n");
  printf("TO UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY.\n");

  /* initialize data structures */
  dbHistoryOpenParam.piDatabaseAlias = dbAlias;
  dbHistoryOpenParam.piTimestamp = NULL;
  dbHistoryOpenParam.piObjectName = NULL;
  dbHistoryOpenParam.iCallerAction = DB2HISTORY_LIST_HISTORY;
  dbHistoryEntryGetParam.pioHistData = &histEntryData;
  dbHistoryEntryGetParam.iCallerAction = DB2HISTORY_GET_ALL;
  rc = HistoryEntryDataFieldsAlloc(&histEntryData);
  if (rc != 0)
  {
    return rc;
  }

  /*****************************************/
  /* OPEN THE DATABASE RECOVERY HISTORY FILE */
  /*****************************************/
  printf("\n  Open the recovery history file for '%s' database.\n", dbAlias);

  /* The API db2HistoryOpenScan starts a recovery history file scan */
  db2HistoryOpenScan(db2Version710, &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(db2Version710, &dbHistoryEntryGetParam, &sqlca);
DB2_API_CHECK("first recovery history file entry -- read");
printf("\n  Display the first entry.\n");

/* HistoryEntryDisplay is a support function used to display the entries
   in the recovery history file. */
rc = HistoryEntryDisplay(histEntryData);

/* update the first history file entry */
rc = DbConn(dbAlias, user, pswd);
if (rc != 0)
{
  return rc;
}

strcpy(newLocation, "this is the NEW LOCATION");
strcpy(newComment, "this is the NEW COMMENT");
printf("\n  Update the first entry in the history file:\n");
printf("    new location = '%s'\n", newLocation);
printf("    new comment = '%s'\n", newComment);
dbHistoryUpdateParam.piNewLocation = newLocation;
dbHistoryUpdateParam.piNewDeviceType = NULL;
dbHistoryUpdateParam.piNewComment = newComment;
dbHistoryUpdateParam.iEID.ioNode = histEntryData.oEID.ioNode;
dbHistoryUpdateParam.iEID.ioHID = histEntryData.oEID.ioHID;

/* The API db2HistoryUpdate can be used to update the location,
   device type, or comment in a history file entry. */

/* Call this API to update the location and comment of the first
   entry in the history file: */
db2HistoryUpdate(db2Version710, &dbHistoryUpdateParam, &sqlca);
DB2_API_CHECK("first history file entry -- update");

rc = DbDisconn(dbAlias);
if (rc != 0)
{
  return rc;
}

/*******************************************/
/* CLOSE THE DATABASE RECOVERY HISTORY FILE */
/*******************************************/
printf("\n  Close recovery history file for '%s' database.\n", dbAlias);
```

```
  /* The API db2HistoryCloseScan ends the recovery history file scan and
     frees DB2 resources required for the scan. */
  db2HistoryCloseScan(db2Version710, &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(db2Version710, &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(db2Version710, &dbHistoryEntryGetParam, &sqlca);
  DB2_API_CHECK("first recovery history file entry -- read");

  printf("\n  Display the first entry.\n");
  rc = HistoryEntryDisplay(histEntryData);

  /*******************************************/
  /* CLOSE THE DATABASE RECOVERY HISTORY FILE */
  /*******************************************/
  printf("\n  Close the recovery history file for '%s' database.\n",
         dbAlias);

  /* ends the recovery history file scan */
  db2HistoryCloseScan(db2Version710, &recoveryHistoryFileHandle, &sqlca);
  DB2_API_CHECK("database recovery history file -- close");

  /* free the allocated memory */
  rc = HistoryEntryDataFieldsFree(&histEntryData);

  return 0;
} /* DbFirstRecoveryHistoryFileEntryUpdate */

int DbRecoveryHistoryFilePrune(char dbAlias[], char user[], char pswd[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2PruneStruct histPruneParam;
  char timeStampPart[14 + 1];

  printf("\n*************************************\n");
  printf("*** PRUNE THE RECOVERY HISTORY FILE ***\n");
  printf("*************************************\n");
```

```
  printf("\nUSE THE DB2 API:\n");
  printf("  db2Prune -- Prune Recovery History File\n");
  printf("AND THE SQL STATEMENTS:\n");
  printf("  CONNECT\n");
  printf("  CONNECT RESET\n");
  printf("TO PRUNE THE RECOVERY HISTORY FILE.\n");

  /* Connect to the database: */
  rc = DbConn(dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  /* Prune the recovery history file: */
  printf("\n  Prune the recovery history file for '%s' database.\n",
         dbAlias);

  /* timeStampPart is a pointer to a string specifying a time stamp or
     log sequence number. Time stamp is used here to select records for
     deletion. All entries equal to or less than the time stamp will be
     deleted. */
  histPruneParam.piString = timeStampPart;
  strcpy(timeStampPart, "2010"); /* year 2010 */

  /* The action DB2PRUNE_ACTION_HISTORY removes history file entries: */
  histPruneParam.iAction = DB2PRUNE_ACTION_HISTORY;

  /* The option DB2PRUNE_OPTION_FORCE forces the removal of the last backup: */
  histPruneParam.iOptions = DB2PRUNE_OPTION_FORCE;

  /* db2Prune can be called to delete entries from the recovery history file
     or log files from the active log path. Here we call it to delete
     entries from the recovery history file.
     You must have SYSADM, SYSCTRL, SYSMAINT, or DBADM authority to prune
     the recovery history file. */
  db2Prune(db2Version710, &histPruneParam, &sqlca);
  DB2_API_CHECK("recovery history file -- prune");

  /* Disconnect from the database: */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* DbRecoveryHistoryFilePrune */
```

Appendix F. Recovery CLP Script

The following DB2 command script shows how to use CLP commands to:

- Back up a database
- Restore the database
- Rollforward recover the database

Ensure that the SAMPLE database exists and is not in use. For detailed information about the SAMPLE database, see the *SQL Reference*. For general information about the DB2 command line processor, see the *Command Reference*.

Both a Windows-compatible and a UNIX-compatible version of the script are described.

Sample Command Script for Windows Operating Systems

To run the script on Windows NT or Windows 2000:

1. Save the script to a file named, for example, `backrest.db2`.
2. If the database manager is not running, issue the **db2start** command from a DB2 command window. To open a CLP-enabled DB2 window, and initialize the DB2 command line environment on the Windows operating system, issue **db2cmd** from a command prompt.
3. Enter `db2 -f backrest.db2 -t`.

The following is an example of the output that this script returns:

```
D:\>db2 -f backrest.db2 -t
This is CLP script: backrest.db2


Deleting old SAMPLE database backup images...

process SAMPLE.0\DB2\NODE0000\CATN0000
process 20010403

Updating the database configuration parameter LOGRETAIN to 'ON'...

DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
DB21026I  For most configuration parameters, all applications must disconnect
from this database before the changes become effective.


Backing up the SAMPLE database...
```

```
Backup successful. The timestamp for this backup image is : 20010403131027


Restoring the SAMPLE database as TESTBACK (1st pass)...

SQL1277N  Restore has detected that one or more table space containers are
inaccessible, or has set their state to 'storage must be defined'.
DB20000I  The RESTORE DATABASE command completed successfully.

Listing the table spaces for the TESTBACK database...


           Tablespaces for Current Database

 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x2001100
   Detailed explanation:
     Restore pending
     Storage must be defined
     Storage may be defined

 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = System Temporary data
 State                                = 0x2001100
   Detailed explanation:
     Restore pending
     Storage must be defined
     Storage may be defined

 Tablespace ID                        = 2
 Name                                 = USERSPACE1
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x2001100
   Detailed explanation:
     Restore pending
     Storage must be defined
     Storage may be defined


Defining new table space containers for Tablespace 2...

DB20000I  The SET TABLESPACE CONTAINERS command completed successfully.

Listing table space containers for Tablespace 2 (TESTBACK database)...


            Tablespace Containers for Tablespace 2
```

```
 Container ID                         = 0
 Name                                 = c:\ts2con1
 Type                                 = Path


Restoring the SAMPLE database as TESTBACK (2nd pass)...

DB20000I  The RESTORE DATABASE command completed successfully.

Rolling the TESTBACK database forward...


                               Rollforward Status

 Input database alias                 = testback
 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           = 2001-04-03-03.16.07.000000

DB20000I  The ROLLFORWARD command completed successfully.

Dropping the TESTBACK database...

DB20000I  The DROP DATABASE command completed successfully.

Terminating the command line processor's back-end process...

DB20000I  The TERMINATE command completed successfully.


D:\>
```

Following is the source listing for the script:

```
-- Before proceeding, ensure that:
--   The database manager is running
--   The SAMPLE database exists and is not in use.

-- Run the script by issuing:
--   db2 -f backrest.db2 -t
--     where -f tells the command line processor to read command input
--              from a file instead of from standard input, and
--           -t tells the command line processor to use a semicolon (;)
--              as the statement termination character.

!ECHO This is CLP script: backrest.db2;

-- Ensure that the DB2 profile registry variable DB2_ENABLE_LDAP
--   is set to 'NO':
!db2set DB2_ENABLE_LDAP=NO;
```

Sample Command Script for Windows Operating Systems

```
!ECHO Deleting old SAMPLE database backup images...;
!rd! SAMPLE.0\DB2\NODE0000\CATN0000;

!ECHO Updating the database configuration parameter LOGRETAIN to 'ON'...;
update db cfg for sample using logretain on;

!ECHO Backing up the SAMPLE database...;
backup db sample;

!ECHO Restoring the SAMPLE database as TESTBACK (1st pass)...;
restore db sample into testback redirect;

!ECHO Listing the table spaces for the TESTBACK database...;
list tablespaces;

!ECHO Defining new table space containers for Tablespace 2...;
set tablespace containers for 2 using (path "c:\ts2con1");

!ECHO Listing table space containers for Tablespace 2 (TESTBACK database)...;
list tablespace containers for 2;

!ECHO Restoring the SAMPLE database as TESTBACK (2nd pass)...;
restore db sample continue;

!ECHO Rolling the TESTBACK database forward...;
rollforward db testback stop;

!ECHO Dropping the TESTBACK database...;
drop db testback;

!ECHO Terminating the command line processor's back-end process...;
terminate;

-- End file
```

Sample Command Script for UNIX Based Systems

To run the script on a UNIX based operating system:

1. Save the script to a file named, for example, `backrest.db2`.
2. If the database manager is not running, issue the **db2start** command from a command prompt.
3. Enter `db2 -f backrest.db2 -t`.

The following is an example of the output that this script returns:

```
sunfish /export/home2/falexand/samples/clp>db2 -f backrest.db2 -t
This is CLP script: backrest.db2

Deleting old SAMPLE database backup images...

Updating the database configuration parameter LOGRETAIN to 'ON'...
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
```

```
DB21026I  For most configuration parameters, all applications must disconnect
from this database before the changes become effective.


Backing up the SAMPLE database...

Backup successful. The timestamp for this backup image is : 20010531172525


Restoring the SAMPLE database as TESTBACK (1st pass)...
SQL1277N  Restore has detected that one or more table space containers are
inaccessible, or has set their state to 'storage must be defined'.
DB20000I  The RESTORE DATABASE command completed successfully.

Listing the table spaces for the TESTBACK database...

           Tablespaces for Current Database

 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x2001100
   Detailed explanation:
     Restore pending
     Storage must be defined
     Storage may be defined

 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = System Temporary data
 State                                = 0x2001100
   Detailed explanation:
     Restore pending
     Storage must be defined
     Storage may be defined

 Tablespace ID                        = 2
 Name                                 = USERSPACE1
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x2001100
   Detailed explanation:
     Restore pending
     Storage must be defined
     Storage may be defined


Defining new table space containers for Tablespace 2...
DB20000I  The SET TABLESPACE CONTAINERS command completed successfully.

Listing table space containers for Tablespace 2 (TESTBACK database)...

            Tablespace Containers for Tablespace 2
```

```
 Container ID                         = 0
 Name                                 = /export/home2/falexand/falexand...
                                          .../NODE0000/SQL00020/ts2con1
 Type                                 = Path


Restoring the SAMPLE database as TESTBACK (2nd pass)...
DB20000I  The RESTORE DATABASE command completed successfully.

Rolling the TESTBACK database forward...

                               Rollforward Status

 Input database alias                 = testback
 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           = 2001-05-29-21.20.16.000000

DB20000I  The ROLLFORWARD command completed successfully.

Dropping the TESTBACK database...
DB20000I  The DROP DATABASE command completed successfully.

Terminating the command line processor's back-end process...
DB20000I  The TERMINATE command completed successfully.
```

Following is the source listing for the script:

```
-- Before proceeding, ensure that:
--   The database manager is running
--   The SAMPLE database exists and is not in use.

-- Run the script by issuing:
--   db2 -f backrest.db2 -t
--     where -f tells the command line processor to read command input
--               from a file instead of from standard input, and
--           -t tells the command line processor to use a semicolon (;)
--              as the statement termination character.

echo This is CLP script: backrest.db2;

-- Ensure that the DB2 profile registry variable DB2_ENABLE_LDAP
--   is set to 'NO':
!db2set DB2_ENABLE_LDAP=NO;

echo Deleting old SAMPLE database backup images...;
!rm -f ./SAMPLE.0.*;

echo Updating the database configuration parameter LOGRETAIN to 'ON'...;
```

```
update db cfg for sample using logretain on;

echo Backing up the SAMPLE database...;
backup db sample;

echo Restoring the SAMPLE database as TESTBACK (1st pass)...;
restore db sample into testback redirect;

echo Listing the table spaces for the TESTBACK database...;
list tablespaces;

echo Defining new table space containers for Tablespace 2...;
set tablespace containers for 2 using (path "ts2con1");

echo Listing table space containers for Tablespace 2 (TESTBACK database)...;
list tablespace containers for 2;

echo Restoring the SAMPLE database as TESTBACK (2nd pass)...;
restore db sample continue;

echo Rolling the TESTBACK database forward...;
rollforward db testback stop;

echo Dropping the TESTBACK database...;
drop db testback;

echo Terminating the command line processor's back-end process...;
terminate;

-- End file
```

Sample Command Script for UNIX Based Systems

Appendix G. Tivoli Storage Manager

When invoking the DB2 backup or the restore utility, you can specify that you want to use the Tivoli Storage Manager (TSM, formerly ADSM) product to manage the backup or the restore operation.. You can use TSM client Version 3.1.x.3 and later with DB2.

Setting up a Tivoli Storage Manager Client on UNIX Based Platforms

Before the database manager can use the TSM option, the following setup activities must be performed:

1. On SunOS and Solaris environments, perform the following steps. (For other UNIX based platforms, begin at step 2.)
   a. Ensure that the required operating system level is installed: SunOS 5.5.1 or Solaris 2.5.1.
   b. Install the TSM client Version 3.1.x.3 or later. Ensure that you remove *all* previous TSM packages before installing this version of the client.
   c. Verify that TSM is installed in the directories `/opt/IBMDSMap5`, `/opt/IBMDSMba5`, and `/opt/IBMDSMsa5`.
   d. Create the following symbolic links in the directory `/usr/lib`, if they do not already exist:

      ```
      libApiDS.so -> libApiDS.so.1
      libApiDS.so.1 -> /opt/IBMDSMap5/api/libApiDS.so.2
      ```

2. Create or modify the TSM user configuration options file `/usr/sbin/dsm.opt`, and the TSM system configuration options file `/usr/sbin/dsm.sys` to suit your environment.
3. On SunOS and Solaris environments, perform the following steps. (For other UNIX based platforms, continue at step 4.)
   a. Copy `/usr/sbin/dsm.opt` and `/usr/sbin/dsm.sys` to the directory `/opt/IBMDSMap5`.
   b. Copy `/opt/IBMDSMap5/solaris/dsmaptica` to the directory `/opt/IBMDSMap5`.
4. Set the environment variables used by TSM:

   DSMI_DIR Identifies the user-defined directory path where the API trusted agent file (`dsmapicta` or `dsmtca`) is located. On SunOS and Solaris environments, this should be set to `/opt/IBMDSMap5`.

   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. On SunOS and Solaris environments, this should be set to `/opt/IBMDSMap5/dsm.opt`.

DSMI_LOG Identifies the user-defined directory path where the error log (`dsierror.log`) will be created.

5. Establish the TSM password.

   For a Tivoli client to be able to interface with a TSM server, it must have a password for the server. The executable file `dsmapipw` is installed in the `INSTHOME/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 "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 that you execute this command, the 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. (Note that you will be prompted twice for the new password, to check for input errors.)

   Note: Users invoking the backup or the restore utility 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.

6. If the database manager is running, you should:
   - Stop the database manager using the **db2stop** command.
   - Start the database manager using the **db2start** command.

Setting up a Tivoli Storage Manager Client on Other Platforms

Before the database manager can use the TSM option, the following setup activities must be performed:

1. Set the environment variables used by TSM:

   DSMI_DIR Identifies the user-defined directory path where the API trusted agent file (`dsmapicta` or `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.

2. If applicable to your operating system, create (or modify) the TSM system configuration options file (`dsm.sys`).
3. Create (or modify) the `dsm.opt` TSM user configuration options file. The environment variable `DSMI_CONFIG` points to this file.
4. Establish the TSM password.

   For a Tivoli client to be able to interface with a TSM server, it must have a password for the server. 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. When this command is run, 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 that you invoke this command, the 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. (Note that you will be prompted twice for the new password, to check for input errors.)

   Note: Users invoking the backup or the restore utility 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.
5. If the database manager is running, you should:
   - Stop the database manager using the **db2stop** command.
   - Start the database manager using the **db2start** command.

Considerations for Using Tivoli Storage Manager

To use specific features within TSM, you may be required to give the fully qualified path name of the object using the feature. (Remember that on the Windows NT operating system and OS/2, \ will be used instead of /.) The fully qualified path name of:

- A full database backup object is:
  `/<database>/NODEnnnn/FULL_BACKUP.timestamp.seq_no`
- A table space backup object is:
  `/<database>/NODEnnnn/TSP_BACKUP.timestamp.seq_no`
- A load copy object is: `/<database>/NODEnnnn/LOAD_COPY.timestamp.seq_no`

where <database> is the database alias name, and NODEnnnn is the node number. The names shown in uppercase characters must be entered as shown.

- In the case where you have multiple backup images using the same database alias name, the time stamp and sequence number become the distinguishing part of a fully qualified name. You will need to query TSM to determine which backup version to use.
- Individual backup images are not known to the TSM graphical user interface. 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 (see "db2adutl - Work with TSM Archived Images" on page 302).
- The TSM server will time out a session if the Tivoli client does not respond within the period of time specified by the COMMTIMEOUT parameter in the server's configuration file. Three factors can contribute to a timeout problem:
  - The COMMTIMEOUT parameter may be set too low at the TSM server. For example, during a restore operation, a timeout can occur if large DMS table spaces are being created. The recommended value for this parameter is 6000 seconds.
  - The DB2 backup or restore buffer may be too large.
  - Database activity during an online backup operation may be too high.
- The database manager uses the TSM full backup option; TSM incremental backup operations are not supported.
- Use multiple sessions to increase throughput.
- On non-UNIX based systems, the DB2 backup and restore utilities do not allow more than one TSM session.

Current Tivoli clients on Windows operating systems and OS/2 support reentrancy, and so multiple I/O sessions can safely be created with the backup, restore, or load utilities from a single machine. However, users must confirm that their installed version of the TSM client supports this function.

In a single-node configuration, if a user attempts to issue a BACKUP DATABASE command such as:

```
db2 backup db sample use tsm open 3 sessions
```

DB2 will detect that multiple sessions are not supported by TSM, and will return an error message. (This also applies to load operations invoked with the COPY YES option, using TSM.)

Be aware, however, that in a multiple logical node (MLN) configuration on Windows NT, DB2 may not be able to detect the use of multiple sessions on a single machine if each logical node attempts to create only one session. For this reason it is very important for MLN configurations to verify that their

TSM client supports reentrancy. If multiple logical nodes are being backed up, restored, or loaded in parallel using TSM, DB2 will allow the operation to proceed if each node attempts to use a single session, even though the logical nodes actually reside on the same hardware. This can lead to failed backup attempts, and hung load processes, and should not be attempted without the most recent TSM client.

Managing Backups and Log Archives on TSM

The **db2adutl** utility allows you to query, extract, and delete backup images, logs, and load copy images that were saved using TSM. The utility is installed in the `INSTHOME/sqllib/misc` directory on UNIX based systems, and in the `\sqllib\misc` directory on Windows operating systems and on OS/2. For detailed information about this utility, see "db2adutl - Work with TSM Archived Images" on page 302.

The QUERY option allows you to list backup images, load copy images, or logs. You can select a range of logs to be listed. You can also request to see the inactive backup images.

The EXTRACT option allows you to copy backup images or logs from TSM to your current directory. You can select which backup images or logs to extract.

The DELETE option allows you to deactivate backup images or to delete logs from TSM. You can select which backup images or logs to process. You can use the KEEP *n* option to keep the most recent *n* backup images. You can also use the OLDER THAN *timestamp* or *n* DAYS option to tailor the DELETE request.

Tivoli Space Manager Integration with Data Links

DB2 Data Links Manager can use Tivoli Space Manager (TSM) and its virtual file system (FSM), which layers itself on top of the native journaled file system (JFS). FSM can be accessed and configured in the same manner as JFS.

This feature benefits users who have file systems with large files that must periodically be moved to tertiary storage. The space for these file systems must be managed on a regular basis. DB2 Data Links Manager support of TSM provides greater flexibility in managing the space for DATALINK files. Rather than pre-allocating enough storage in the DB2 Data Links Manager file system for all files that may be stored there, TSM allows allocations of the Data Links managed file system to be adjusted over a period of time without the risk of inadvertently filling up the file system during normal usage.

Restrictions and Limitations

- This feature is currently supported on AIX only.

- Selective migration (**dsmmigrate**) and recall of an FC (read permission database) linked file should be done by a root user only.

  Selective migration can only be performed by the file owner which, in the case of read permission database files, is the DataLink Manager Administrator (dlfm). To access such files, a token is required from the host database. The only user who does not require a token is the "root" user. It is easier for a "root" user to perform selective migration and recall on such files. The dlfm user can only migrate an FC file using a valid token the first time. The second time that migration is attempted (after a recall), the operation fails with error message "ANS1028S Internal program error. Please see your service representative." If a non-root user tries to run **dsmmigrate** on an FC file, the operation does not succeed. This limitation is minor, because it is typically administrators who access files on the file server.
- The **stat** and **statfs** system calls show *Vfs-type* as `fsm` rather than `dlfs`, even though dlfs is mounted over fsm.

  This behavior supports the normal functionality of **dsmrecalld** daemons, which run **statfs** on the file system to determine whether or not its *Vfs-type* is `fsm`.
- The **dsmls** command does not show any output if a file having the minimum *inode* number is FC (read permission database) linked.

  The **dsmls** command is similar to the **ls** command: it lists the files being administered by TSM. No user action is required.

Appendix H. User Exit for Database Recovery

You can develop a *user exit program* to automate log file archiving and retrieval. (On OS/2, user exit programs can also be used for backup and restore operations.) Before invoking a user exit program for log file archiving or retrieval, ensure that the *userexit* database configuration parameter has been set to `YES`. This also enables your database for rollforward recovery.

When a user exit program is invoked, the database manager passes control to the executable file, `db2uext2`. (On OS/2, backup and restore operations call `db2uexit.cmd` first, which in turn calls `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 444). And because only one user exit program can be invoked within a database manager instance, it must have a section for each of the operations it may be asked to perform.

The following topics are covered:

Sample User Exit Programs

Sample user exit programs are provided for all supported platforms. You can modify these programs to suit your particular requirements. The sample programs are well commented with information that will help you to use them most effectively.

You should be aware that user exit programs must *copy* log files from the active log path to the archive log path. Do not remove log files from the active log path. (This could cause problems during database recovery.) DB2 removes archived log files from the active log path when these log files are no longer needed for recovery.

Following is a description of the sample user exit programs that are shipped with DB2.

- **UNIX based systems**

The user exit sample programs for DB2 for UNIX based systems are found in the sqllib/samples/c subdirectory. Although the samples provided are coded in C, your user exit program can be written in a different programming language.

Your user exit program must be an executable file whose name is db2uext2.

There are four sample user exit programs for UNIX based systems:

– db2uext2.cadsm

  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 uses the Legato NetWorker** Version 4.2.5 program, available from Legato** Systems, Inc. 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.cadsm

    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.

- **OS/2**

  The user exit sample programs for DB2 for OS/2 are found in the instance subdirectory of the \sqllib\samples\rexx directory. (The dbuexit.CAD program is an exception: it is found in the instance subdirectory of the \sqllib\samples\c directory.) Although the samples provided are mostly REXX command files, your user exit program can be written in a different programming language.

  The sample that you choose to implement should be renamed db2uexit, with an extension of either .cmd or .exe. Move the renamed file to the \sqllib\bin directory.

There are five OS/2 sample user exit programs:

– `db2uexit.ex1`

  This sample uses the Sytos Premium** Version 2.2 program, available from Seagate** Software Inc. It can be used to store data on, and retrieve data from, an IBM external tape device. Only Version 2.2 of the Sytos Premium product is currently supported. (You need OS/2 FixPak 26 to use this product.) Review the sample program listing to learn about other requirements.

– `db2uexit.ex2`

  This sample uses the Filesafe** program, available from the Mountain** Corporation. It can be used to store data on, and retrieve data from, a Mountain tape device. A unique volume label is assigned to each backup copy of a database, so that multiple backup images from one or more databases can be stored on the same tape.

– `db2uexit.ex3`

  This sample uses the MaynStream** program, available from the Maynard** Corporation. It can be used to store data on, and retrieve data from, a Maynard tape device. MaynStream does not support redirecting the database restore operation to a drive other than the one on which the database was backed up.

– `db2uexit.ex4`

  This sample uses the OS/2 XCOPY command. The storage device can be any device supported by OS/2, such as a fixed disk, diskette, or optical cartridge. These devices can be LAN-redirected drives if the workstation is configured appropriately.

  XCOPY cannot be used for backing up and restoring databases.

– `db2uexit.CAD`

  This sample, written in C, is equivalent to the Tivoli Storage Manager (TSM) sample program. It can be used 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.

- Calling Format for UNIX Based Operating Systems or Windows NT/2000:

```
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`, `Dynix/ptx`, `SGI`, and `NT`.

Calling Format

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.

- Calling Format for OS/2:

```
action drive db_alias log_path log_file indicator
```

action Valid values are: `BACKUP`, `RESTORE`, `ARCHIVE`, and `RETRIEVE`.

drive For a backup operation, specifies the drive on which the database that is to be backed up is located. For a restore operation, specifies the drive to which the database is to be restored. For an archiving or a retrieval operation, specifies the drive on which the database is located. In each case, specify a drive letter followed by a colon (for example, `C:`).

db_alias Specifies the database alias (or the database name, if no alias exists).

log_path For a backup or a restore operation, specifies the fully qualified name of a response file, which contains a list of files to be backed up or restored. Each name in the list is a fully qualified file name that may contain wild card characters. For restore operations, the drive letter and the path represent the source drive and the path for the database file when it was backed up. For example, if `C:\SQLUTIL\dbname.MH1` is contained in the response file, it means that the `dbname.MH1` file was backed up from `C:\SQLUTIL`.

For an archiving or a retrieval operation, specifies the log path directory. For example, `C:\SQL00001\SQLOGDIR\`.

log_file For a backup operation, specifies a media label that was generated by the backup utility. This label is composed of the database alias name and a time stamp. For a restore operation, specifies the path name of the database subdirectory to which the files are to be restored. The drive letter is not included, because it is specified through the *drive* parameter. The format is `\SQL`*nnnnn*`\`.

For an archiving or a retrieval operation, specifies the log file name. For example, `S0000001.LOG`.

indicator Specifies an indicator that can be used to support multiple calls during a backup or a restore operation. The first call has a character value of `1`, and subsequent calls have a character value of `2`.

The user exit program is called multiple times during a backup or a restore operation. The first call backs up or restores media header (`.MH`*n*) files, and the second call backs up or restores the entire set of database files.

This parameter is not used for archiving or retrieval operations.

Backup and Restore Considerations (DB2 for OS/2 only)

The following considerations apply if you are writing a user exit program that is called from the backup or the restore utility:

- A non-zero return code returned by a user exit program causes the utility to fail, and no retry is attempted.
- Use only supported wild card characters in fully qualified file names. For example, `C:\SQL00001\*.*` and `C:\*.MH*` are both acceptable search criteria.
- The user exit program must handle the response file format of one fully qualified file name per line, with each line terminated by a carriage return and line feed character. There is no end-of-file character in the file.
- If multiple backup images for the same database are to be placed on one medium, the user exit program should be able to select the correct image during a restore operation. (See the description of `db2uexit.ex2` in "Sample User Exit Programs" on page 439.)
- Two concurrently running backup operations that are sharing one backup device must be serialized.
- If a backup image spans more than one medium, the prompting for media must be handled by the user exit program, or by an application that it calls.

To support this feature, the backup and the restore utilities open an operating system foreground session to call the user exit program.

- The user exit program must not back up any subdirectory within the database directory.
- When restoring a database with a user exit program, the restore utility requires complete control over that database. However, the workstation can have active connections to databases other than the one being restored.
- If a database is being backed up or restored with a user exit program, and another operation is using the same tape device, the backup or restore operation may fail, and will need to be restarted. To avoid this situation, you can ensure that no other databases that call the user exit program for logging are in use while a backup or a restore operation is in progress, or you can ensure that the user exit program retries the backup or the restore operation at a later time if a device is not ready.
- During a restore operation, the drive letter and the path can be different than those that were specified during the backup operation. For example, if file `dbname.MH1` is backed up from `C:\SQLUTIL`, you can restore it to `D:\SQLUTIL2`.

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.

On OS/2, any non-zero code returned by a user exit program causes the backup or the restore utility to fail, and no retry is attempted. The utilities report a general SQLCODE -2029, whose message text displays the code returned by the user exit program or the operating system.

Table 24 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 24. 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]

Table 24. User Exit Program Return Codes (continued). Applies to archiving and retrieval operations only.

Return Code	Explanation
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. On OS/2, this error message can also mean that a file needed to complete a restore operation could not be found on the current backup media.[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 24. 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 hangs. (This applies to rollforward operations, or calls to the **sqlurlog** API, which is used by the replication utility.) [b] User exit requests are suspended for five minutes. During this time, all requests are ignored, including the request that caused the error condition. Following this five-minute suspension, the next request is processed. If this request is processed without error, processing of new user exit requests continues, and DB2 reissues the archive request that failed or was suspended previously. If a return code greater than 8 is generated during the retry, requests are suspended for an additional five minutes. The five-minute suspensions continue until the problem is corrected, or the database is stopped and restarted. Once all applications have disconnected from the database, DB2 issues an archive request for any log file that may not have been successfully archived previously. If the user exit program fails to archive log files, your disk may become filled with log files, and performance may be degraded. Once the disk becomes full, the database manager will not accept further application requests for database updates. If the user exit program was called to retrieve log files, rollforward recovery is suspended, but not stopped, unless the ROLLFORWARD STOP option was specified. If the STOP option was not specified, you can correct the problem and resume recovery. [c] If the user exit program returns error code 255, it is likely that the program cannot load the library file for the executable. To verify this, manually invoke the user exit program. More information is displayed. **Note:** During archiving and retrieval operations, an alert message is issued for all return codes except 0, 4, and 24. The alert message contains the return code from the user exit program, and a copy of the input parameters that were provided to the user exit program.	

Appendix I. Backup and Restore APIs for Vendor Products

DB2 provides interfaces that can be used by third-party media management products to store and retrieve data for backup and restore operations. This function is designed to augment the backup and restore data targets of diskette, disk, tape, and Tivoli Storage Manager, that are supported as a standard part of DB2.

These third-party media management products will be referred to as vendor products in the remainder of this appendix.

DB2 defines a set of function prototypes that provide a general purpose data interface to backup and restore that can be used by many vendors. These functions are to be provided by the vendor in a shared library on UNIX based systems, or DLL on OS/2 or the Windows operating system. When the functions are invoked by DB2, the shared library or DLL specified by the calling backup or restore routine is loaded and the functions provided by the vendor are called to perform the required tasks.

This appendix is divided into four parts:

- Operational overview of DB2's interaction with vendor products.
- Detailed descriptions of DB2's vendor APIs.
- Information on the data structures used in the API calls.
- Details on invoking backup and restore using vendor products.

Operational Overview

Five functions are defined to interface DB2 and the vendor product:

- sqluvint - Initialize and Link to Device
- sqluvget - Reading Data from Device
- sqluvput - Writing Data to Device
- sqluvend - Unlink the Device
- sqluvdel - Delete Committed Session

DB2 will call these functions, and they should be provided by the vendor product in a shared library on UNIX based systems, or in a DLL on OS/2 or the Windows operating system.

Note: The shared library or DLL code will be run as part of the database engine code. Therefore, it must be reentrant and thoroughly debugged. An errant function may compromise data integrity of the database.

The sequence of functions that DB2 will call during a specific backup or restore operation depends on:

- The number of sessions that will be utilized.
- Whether it is a backup or a restore operation.
- The PROMPTING mode that is specified on the backup or restore operation.
- The characteristics of the device on which the data is stored.
- The errors that may be encountered during the operation.

Number of Sessions

DB2 supports the backup and restore of database objects using one or more data streams or sessions. A backup or restore using three sessions would require three physical or logical devices to be available. When vendor device support is being used, it is the vendor's functions that are responsible for managing the interface to each physical or logical device. DB2 simply sends or receives data buffers to or from the vendor provided functions.

The number of sessions to be used is specified as a parameter by the application that calls the backup or restore database function. This value is provided in the INIT-INPUT structure used by **sqluvint** (see "sqluvint - Initialize and Link to Device" on page 456).

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. For more details, see "INIT-INPUT" on page 474.

When the backup operation completes successfully, DB2 writes a media trailer to the last session it closes. This trailer includes information that tells DB2

how many sessions were used to perform the backup operation. During a restore operation, this information is used to ensure all the sessions, or data streams, have been restored.

Operation with No Errors, Warnings or Prompting

For backup, the following sequence of calls is issued by DB2 for *each* session.

```
sqluvint, action = SQLUV_WRITE
```

followed by 1 to n

```
sqluvput
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

When DB2 issues an **sqluvend** call (action SQLUV_COMMIT), it expects the vendor product to appropriately save the output data. A return code of SQLUV_OK to DB2 indicates success.

The DB2-INFO structure, used on the **sqluvint** call, contains the information required to identify the backup (see "DB2-INFO" on page 470). A sequence number is supplied. The vendor product may choose to save this information. DB2 will use it during restore to identify the backup that will be restored.

For restore, the sequence of calls for each session is:

```
sqluvint, action = SQLUV_READ
```

followed by 1 to n

```
sqluvget
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

The information in the DB2-INFO structure used on the **sqluvint** call will contain the information required to identify the backup. A sequence number is not supplied. DB2 expects that all backup objects (session outputs committed during a backup) will be returned. The first backup object returned is the object generated with sequence number 1, and all other objects are restored in no specific order. DB2 checks the media tail to ensure that all objects have been processed.

Note: Not all vendor products will keep a record of the names of the backup objects. This is most likely when the backups are being done to tapes, or other media of limited capacity. During the initialization of restore sessions, the identification information can be utilized to stage the necessary backup objects so that they are available when required; this

may be most useful when juke boxes or robotic systems are used to store the backups. DB2 will always check the media header (first record in each session's output) to ensure that the correct data is being restored.

PROMPTING Mode

When a backup or a restore operation is initiated, two prompting modes are possible:

- WITHOUT PROMPTING or NOINTERRUPT, where there is no opportunity for the vendor product to write messages to the user, or for the user to respond to them.
- PROMPTING or INTERRUPT, where the user can receive and respond to messages from the vendor product.

For PROMPTING mode, backup and restore define three possible user responses:

- Continue

  The operation of reading or writing data to the device will resume.
- Device terminate

  The device will receive no additional data, and the session is terminated.
- Terminate

  The entire backup or restore operation is terminated.

The use of the PROMPTING and WITHOUT PROMPTING modes is discussed in the sections that follow.

Device Characteristics

For purposes of the vendor device support APIs, two general types of devices are defined:

- Limited capacity devices requiring user action to change the media; for example, a tape drive, diskette, or CDROM drive.
- Very large capacity devices, where normal operations do not require the user to handle media; for example, a juke box, or an intelligent robotic media handling device.

A limited capacity device may require that the user be prompted to load additional media during the backup or restore operation. Generally DB2 is not sensitive to the order in which the media is loaded for either backup or restore operations. It also provides facilities to pass vendor media handling messages to the user. This prompting requires that the backup or restore operation be initiated with PROMPTING on. The media handling message text is specified in the description field of the return code structure.

If PROMPTING is on, and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a **sqluvput** (write) or a **sqluvget** (read) call, DB2:

- Marks the last buffer sent to the session to be resent, if the call was **sqluvput**. It will be put to a session later.
- Calls the session with **sqluvend** (action = SQLUV_COMMIT). If successful (SQLUV_OK return code), DB2:
  - Sends a vendor media handling message to the user from the return code structure that signaled the end-of-media condition.
  - Prompts the user for a continue, device terminate, or terminate response.
- If the response is *continue*, DB2 initializes another session using the **sqluvint** call, and if successful, begins writing data to or reading data from the session. To uniquely identify the session when writing, DB2 increments the sequence number. The sequence number is available in the DB2-INFO structure used with **sqluvint**, and is in the media header record, which is the first data record sent to the session.

  DB2 will not start more sessions than requested when a backup or a restore operation is started, or indicated by the vendor product with a SQLUV_MAX_LINK_GRANT warning on an **sqluvint** call.
- If the response is *device terminate*, DB2 does not attempt to initialize another session, and the number of active sessions is reduced by one. DB2 does not allow all sessions to be terminated by device terminate responses; at least one session must be kept active until the backup or the restore operation completes.
- If the response is *terminate*, DB2 terminates the backup or the restore operation. For more information on exactly what DB2 does to terminate the sessions, see "If Error Conditions Are Returned to DB2" on page 452.

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.

Recovery History File

The recovery 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. For more information about the recovery history file, see "Understanding the Recovery History File" on page 48 . 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

Operational Hints and Tips

For information about the layout of the file, see "Data Structure: db2HistData" on page 352.

When a backup operation completes, one or more records is written to the file. If the output of the backup operation was directed to vendor devices, the DEVICE field in the history record contains a 0, and the LOCATION field contains either:

- The vendor file name specified when the backup operation was invoked.
- The name of the shared library, if no vendor file name was specified.

For more information about specifying this option, see "Invoking a Backup or a Restore Operation Using Vendor Products" on page 479.

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 recovery history file can be used to help locate a backup image if a recovery operation becomes necessary.

Functions and Data Structures

The following sections describe the generic functions and data structures available for use by vendor products.

The APIs for vendor products are:

- "sqluvint - Initialize and Link to Device" on page 456
- "sqluvget - Reading Data from Device" on page 460
- "sqluvput - Writing Data to Device" on page 463
- "sqluvend - Unlink the Device and Release its Resources" on page 466
- "sqluvdel - Delete Committed Session" on page 468

The data structures used by the vendor APIs are:

"DB2-INFO" on page 470
Contains information that identifies DB2 to the vendor device.

"VENDOR-INFO" on page 473
Contains information that identifies the vendor and version of the device.

"INIT-INPUT" on page 474
Sets up a logical link between DB2 and the vendor device.

"INIT-OUTPUT" on page 476
Contains output from the device.

"DATA" on page 477
Contains data that was transferred between DB2 and the vendor device.

"RETURN-CODE" on page 478
Contains a return code and an explanation of the error.

sqluvint - Initialize and Link to Device

This function is called to provide information for initialization and establishment of a logical link between DB2 and the vendor device.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

Init_input
Input. Structure that contains information provided by DB2 to establish a logical link with the vendor device.

Init_output
Output. Structure that contains the output returned by the vendor device.

Return_code
Output. Structure that contains the return code to be passed to DB2, and a brief text explanation.

Usage Notes

For each media I/O session, DB2 will call this function to obtain a device handle. If for any reason, the vendor function encounters an error during initialization, it will indicate it via a return code. If the return code indicates an error, DB2 may choose to terminate the operation by calling the **sqluvend** function. Details on possible return codes, and the DB2 reaction to each of these, is contained in the return codes table (see Table 25 on page 457).

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 (see "PROMPTING Mode" on page 450 and "Device Characteristics" on page 450 for more information).

DB2 names the backup being written or the restore to be read via fields in the DB2-INFO structure. In the case of an action = SQLUV_READ, the vendor product must check for the existence of the named object. If it cannot be found, the return code should be set to SQLUV_OBJ_NOT_FOUND so that DB2 will take the appropriate action.

After initialization is completed successfully, DB2 will continue by issuing other data transfer functions, but may terminate the session at any time with an **sqluvend** call.

Return Codes

Table 25. Valid Return Codes for sqluvint and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput, sqluvget (see comments)	If action = SQLUV_WRITE, the next call will be sqluvput (to BACKUP data). If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_OK; the next call will be sqluvget to RESTORE data.
SQLUV_LINK_EXIST	Session activated previously.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

Table 25. 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 call will not be received, since the session was never established.
SQLUV_INV_VERSION	The DB2 and vendor products are incompatible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ACTION	Invalid action is requested. This could also be used to indicate that the combination of parameters results in an operation which is not possible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NO_DEV_ AVAIL	No device is available for use at the moment.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJ_NOT_ FOUND	Object specified cannot be found. This should be used when the action on the sqluvint call is 'R' (read) and the requested object cannot be found based on the criteria specified in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJS_FOUND	More than 1 object matches the specified criteria. This will result when the action on the sqluvint call is 'R' (read) and more than one object matches the criteria in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_USERID	Invalid userid specified.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ PASSWORD	Invalid password provided.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_OPTIONS	Invalid options encountered in the vendor options field.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INIT_FAILED	Initialization failed and the session is to be terminated.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_DEV_ERROR	Device error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

Table 25. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_MAX_LINK_ GRANT	Max number of links established.	sqluvput, sqluvget (see comments)	This is treated as a warning by DB2. The warning tells DB2 not to open additional sessions with the vendor product, because the maximum number of sessions it can support has been reached (note: this could be due to device availability). If action = SQLUV_WRITE (BACKUP), the next call will be sqluvput. If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_MAX_LINK_GRANT; the next call will be sqluvget to RESTORE data.
SQLUV_IO_ERROR	I/O error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NOT_ ENOUGH_SPACE	There is not enough space to store the entire backup image; the size estimate is provided as a 64-bit value in bytes.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

sqluvget - Reading Data from Device

After initialization, this function can be called to read data from the device.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sqluvend.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data         *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API Parameters

pVendorCB
Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data Input/output. A pointer to the *data* structure.

Return_code
Output. The return code from the API call.

obj_num
Specifies which backup object should be retrieved.

buff_size
Specifies the buffer size to be used.

actual_buff_size
Specifies the actual bytes read or written. This value should be set to output to indicate how many bytes of data were actually read.

dataptr
A pointer to the data buffer.

reserve
Reserved for future use.

Usage Notes

This function is used by the restore utility.

Return Codes

Table 26. Valid Return Codes for sqluvget and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvget	DB2 processes the data
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA or SQLUV_ENDOFMEDIA_NO_DATA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvget, or sqluvend, action = SQLU_ABORT	See the explanation of DB2's handling of warnings in "Warning Conditions" on page 453.
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_MORE_DATA	Operation successful; more data available.	sqluvget	
SQLUV_ENDOFMEDIA_NO_DATA	End of media and 0 bytes read (for example, end of tape).	sqluvend	See the explanation of DB2's handling of end-of-media conditions under "PROMPTING Mode" on page 450, and "Device Characteristics" on page 450.
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 as described under "PROMPTING Mode" on page 450, and "Device Characteristics" on page 450.

Table 26. Valid Return Codes for sqluvget and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
Next call: [a] If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated.			

sqluvput - Writing Data to Device

After initialization, this function can be used to write data to the device.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sqluvend.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API Parameters

pVendorCB
Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data Output. Data buffer filled with data to be written out.

Return_code
Output. The return code from the API call.

obj_num
Specifies which backup object should be retrieved.

buff_size
Specifies the buffer size to be used.

actual_buff_size
Specifies the actual bytes read or written. This value should be set to indicate how many bytes of data were actually read.

dataptr
A pointer to the data buffer.

reserve
Reserved for future use.

Usage Notes

This function is used by the backup utility.

Return Codes

Table 27. Valid Return Codes for sqluvput and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput or sqluvend, if complete (for example, DB2 has no more data)	Inform other processes of successful operation.
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_ENDOFMEDIA	End of media reached, for example, end of tape.	sqluvend	See the explanation of DB2's handling of end-of-media conditions under "PROMPTING Mode" on page 450, and "Device Characteristics" on page 450.
SQLUV_DATA_RESEND	Device requested to have buffer sent again.	sqluvput	DB2 will retransmit the last buffer. This will only be done once.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvput	See the explanation of DB2's handling of warnings in "Warning Conditions" on page 453.
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.

Table 27. Valid Return Codes for sqluvput and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
Next call: [a] If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated. Committed sessions are deleted with an sqluvint, sqluvdel, and sqluvend sequence of calls (see "If Error Conditions Are Returned to DB2" on page 452).			

sqluvend - Unlink the Device and Release its Resources

Ends or unlinks the device, and frees all of its related resources. The vendor must free or release unused resources (for example, allocated space and file handles) before returning to DB2.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  sqlint32 action,
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

action Input. Used to commit or abort the session:

- SQLUV_COMMIT (0 = to commit)
- SQLUV_ABORT (1 = to abort)

pVendorCB
Input. Pointer to the Init_output structure.

Init_output
Output. Space for Init_output de-allocated. The data has been committed to stable storage for a backup if action is to commit. The data is purged for a backup if the action is to abort.

Return code
Output. The return code from the API call.

Usage Notes

This function is called for each session that has been opened. There are two possible action codes:

- Commit

Output of data to this session, or the reading of data from the session, is complete.

For a write (backup) session, if the vendor returns to DB2 with a return code of SQLUV_OK, DB2 assumes that the output data has been appropriately saved by the vendor product, and can be accessed if referenced in a later **sqluvint** call.

For a read (restore) session, if the vendor returns to DB2 with a return code of SQLUV_OK, the data should not be deleted, because it may be needed again.

If the vendor returns SQLUV_COMMIT_FAILED, DB2 assumes that there are problems with the entire backup or restore operation. All active sessions are terminated by **sqluvend** calls with action = SQLUV_ABORT. For a backup operation, committed sessions receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls (see "If Error Conditions Are Returned to DB2" on page 452).

- 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 (see "If Error Conditions Are Returned to DB2" on page 452).

  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 28. Valid Return Codes for sqluvend and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	no further calls	Free all memory allocated for this session and terminate.
SQLUV_COMMIT_FAILED	Commit request failed.	no further calls	Free all memory allocated for this session and terminate.
SQLUV_ABORT_FAILED	Abort request failed.	no further calls	

sqluvdel - Delete Committed Session

Deletes committed sessions.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sqluvend.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

Init_input
Input. Space allocated for Init_input and Return_code.

Return_code
Output. Return code from the API call. The object pointed to by the Init_input structure is deleted.

Usage Notes

If multiple sessions are opened, and some sessions are committed, but one of them fails, this function is called to delete the committed sessions. No sequence number is specified; **sqluvdel** is responsible for finding all of the objects that were created during a particular backup operation, and deleting them. Information in the INIT-INPUT structure is used to identify the output data to be deleted. The call to **sqluvdel** is responsible for establishing any connection or session that is required to delete a backup object from the vendor device. If the return code from this call is SQLUV_DELETE_FAILED, DB2 does not notify the caller, because DB2 returns the first fatal failure and ignores subsequent failures. In this case, for DB2 to have called **sqluvdel**, an initial fatal error must have occurred.

Return Codes

Table 29. Valid Return Codes for sqluvdel and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	no further calls	
SQLUV_DELETE_FAILED	Delete request failed.	no further calls	

DB2-INFO

This structure contains information identifying DB2 to the vendor device.

Table 30. Fields in the DB2-INFO Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
DB2_id	char	An identifier for the DB2 product. Maximum length of the string it points to is 8 characters.
version	char	The current version of the DB2 product. Maximum length of the string it points to is 8 characters.
release	char	The current release of the DB2 product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
level	char	The current level of the DB2 product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
action	char	Specifies the action to be taken. Maximum length of the string it points to is 1 character.
filename	char	The file name used to identify the backup image. If it is NULL, the *server_id*, *db2instance*, *dbname*, and *timestamp* will uniquely identify the backup image. Maximum length of the string it points to is 255 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of the string it points to is 8 characters.
db2instance	char	The db2instance ID. This is the user ID invoking the command. Maximum length of the string it points to is 8 characters.

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.

Table 30. Fields in the DB2-INFO Structure (continued). All fields are NULL-terminated strings.

Field Name	Data Type	Description
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	Lists the objects in the backup image. This is provided to the vendors for their information only.
max_bytes_per_txn	sqlint32	Specifies to the vendor in bytes, the transfer buffer size specified by the user.
image_filename	char	Reserved for future use.
reserve	void	Reserved for future use.
nodename	char	Name of the node at which the backup was generated.
password	char	Password for the node at which the backup was generated.
owner	char	ID of the backup originator.
mcNameP	char	Management class.
nodeNum	SQL_PDB_NODE_TYPE	Node number. Numbers greater than 255 are supported by the vendor interface.

The *filename*, or *server_id*, *db2instance*, *type*, *dbname* and *timestamp* uniquely identifies the backup image. The sequence number, specified by *sequence*, identifies the file extension. When a backup image is to be restored, the same values must be specified to retrieve the backup image. Depending on the vendor product, if *filename* is used, the other parameters may be set to NULL, and vice versa.

Language Syntax

C Structure

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                  *DB2_id;
  char                  *version;
  char                  *release;
  char                  *level;
  char                  *action;
  char                  *filename;
  char                  *server_id;
  char                  *db2instance;
  char                  *type;
  char                  *dbname;
  char                  *alias;
  char                  *timestamp;
  char                  *sequence;
  struct sqlu_gen_list  *obj_list;
  long                  max_bytes_per_txn;
  char                  *image_filename;
  void                  *reserve;
  char                  *nodename;
  char                  *password;
  char                  *owner;
  char                  *mcNameP;
  SQL_PDB_NODE_TYPE     nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

This structure contains information identifying the vendor and version of the device.

Table 31. Fields in the VENDOR-INFO Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
vendor_id	char	An identifier for the vendor. Maximum length of the string it points to is 64 characters.
version	char	The current version of the vendor product. Maximum length of the string it points to is 8 characters.
release	char	The current release of the vendor product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
level	char	The current level of the vendor product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of the string it points to is 8 characters.
max_bytes_per_txn	sqlint32	The maximum supported transfer buffer size. Specified by the vendor, in bytes. This is used only if the return code from the vendor initialize function is SQLUV_BUFF_SIZE, indicating that an invalid buffer size was specified.
num_objects_in_backup	sqlint32	The number of sessions that were used to make a complete backup. This is used to determine when all backup images have been processed during a restore operation.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Vendor_info
{
  char       *vendor_id;
  char       *version;
  char       *release;
  char       *level;
  char       *server_id;
  sqlint32   max_bytes_per_txn;
  sqlint32   num_objects_in_backup;
  void       *reserve;
} Vendor_info;
```

INIT-INPUT

This structure contains information provided by DB2 to set up and to establish a logical link with the vendor device.

Table 32. Fields in the INIT-INPUT Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
DB2_session	struct DB2_info	A description of the session from the perspective of DB2.
size_options	unsigned short	The length of the options field. When using the DB2 backup or restore function, the data in this field is passed directly from the *VendorOptionsSize* parameter.
size_HI_order	sqluint32	High order 32 bits of DB size estimate in bytes; total size is 64 bits.
size_LOW_order	sqluint32	Low order 32 bits of DB size estimate in bytes; total size is 64 bits.
options	void	This information is passed from the application when the backup or the restore function is invoked. This data structure must be flat; in other words, no level of indirection is supported. Byte-reversal is not done, and the code page for this data is not checked. When using the DB2 backup or restore function, the data in this field is passed directly from the *pVendorOptions* parameter.
reserve	void	Reserved for future use.
prompt_lvl	char	Prompting level requested by the user when a backup or a restore operation was invoked. Maximum length of the string it points to is 1 character.
num_sessions	unsigned short	Number of sessions requested by the user when a backup or a restore operation was invoked.

Language Syntax

C Structure

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  sqluint32        size_HI_order;
  sqluint32        size_LOW_order;
  void             *options;
  void             *reserve;
  char             *prompt_lvl;
  unsigned short   num_sessions;
} Init_input;
```

INIT-OUTPUT

This structure contains the output returned by the vendor device.

Table 33. Fields in the INIT-OUTPUT Structure

Field Name	Data Type	Description
vendor_session	struct Vendor_info	Contains information to identify the vendor to DB2.
pVendorCB	void	Vendor control block.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

This structure contains data transferred between DB2 and the vendor device.

Table 34. 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.

Language Syntax

C Structure

```
typedef struct Data
{
  sqlint32  obj_num;
  sqlint32  buff_size;
  sqlint32  actual_buff_size;
  void      *dataptr;
  void      *reserve;
} Data;
```

RETURN-CODE

This structure contains the return code and a short explanation of the error being returned to DB2.

Table 35. Fields in the RETURN-CODE Structure

Field Name	Data Type	Description
return_code[a]	sqlint32	Return code from the vendor function.
description	char	A short description of the return code.
reserve	void	Reserved for future use.

[a] This is a vendor-specific return code that is not the same as the value returned by various DB2 APIs. See the individual API descriptions for the return codes that are accepted from vendor products.

Language Syntax

C Structure

```
typedef struct Return_code
{
  sqlint32  return_code,
  char      description[60],
  void      *reserve,
} Return_code;
```

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 or OS/2).
Number of sessions	Is *Sessions*.
Vendor options	Is not supported.
Vendor file name	Is not supported.
Transfer buffer size	Is (for backup) *Size of each Buffer*, and (for restore) not applicable.

The Command Line Processor (CLP)

The command line processor (CLP) can be used to invoke the DB2 BACKUP DATABASE or the RESTORE DATABASE command.

To specify	The command line processor parameter	
	for backup is	for restore is
Use of vendor device and library name	*library-name*	*shared-library*
Number of sessions	*num-sessions*	*num-sessions*
Vendor options	not supported	not supported
Vendor file name	not supported	not supported
Transfer buffer size	*buffer-size*	*buffer-size*

Application Programming Interface (API)

Two API function calls support backup and restore operations: **sqlubkp** for backup (see "Backup Database API" on page 88, and **sqlurestore** for restore (see "Restore Database API" on page 116.

To specify	The API parameter (for both sqlubkp and sqlurestore) 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*

Appendix J. Using the DB2 Library

The DB2 Universal Database library consists of online help, books (PDF and HTML), and sample programs in HTML format. This section describes the information that is provided, and how you can access it.

To access product information online, you can use the Information Center. For more information, see "Accessing Information with the Information Center" on page 495. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web.

DB2 PDF Files and Printed Books

DB2 Information

The following table divides the DB2 books into four categories:

DB2 Guide and Reference Information
These books contain the common DB2 information for all platforms.

DB2 Installation and Configuration Information
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, Windows, and UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the Application Development Client. The samples are for informational purposes and do not replace the actual programs.

Release notes
These files contain late-breaking information that could not be included in the DB2 books.

The installation manuals, release notes, and tutorials are viewable in HTML directly from the product CD-ROM. Most books are available in HTML on the product CD-ROM for viewing and in Adobe Acrobat (PDF) format on the DB2 publications CD-ROM for viewing and printing. You can also order a printed copy from IBM; see "Ordering the Printed Books" on page 491. The following table lists books that can be ordered.

On OS/2 and Windows platforms, you can install the HTML files under the `sqllib\doc\html` directory. DB2 information is translated into different

languages; however, all the information is not translated into every language. Whenever information is not available in a specific language, the English information is provided

On UNIX platforms, you can install multiple language versions of the HTML files under the `doc/%L/html` directories, where *%L* represents the locale. For more information, refer to the appropriate *Quick Beginnings* book.

You can obtain DB2 books and access information in a variety of ways:

- "Viewing Information Online" on page 494
- "Searching Information Online" on page 498
- "Ordering the Printed Books" on page 491
- "Printing the PDF Books" on page 490

Table 36. DB2 Information

Name	Description	Form Number PDF File Name	HTML Directory
	DB2 Guide and Reference Information		
Administration Guide	*Administration Guide: Planning* provides an overview of database concepts, information about design issues (such as logical and physical database design), and a discussion of high availability.	SC09-2946 db2d1x70	db2d0
	Administration Guide: Implementation provides information on implementation issues such as implementing your design, accessing databases, auditing, backup and recovery.	SC09-2944 db2d2x70	
	Administration Guide: Performance provides information on database environment and application performance evaluation and tuning.	SC09-2945 db2d3x70	
	You can order the three volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8934.		
Administrative API Reference	Describes the DB2 application programming interfaces (APIs) and data structures that you can use to manage your databases. This book also explains how to call APIs from your applications.	SC09-2947 db2b0x70	db2b0

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Application Building Guide	Provides environment setup information and step-by-step instructions about how to compile, link, and run DB2 applications on Windows, OS/2, and UNIX-based platforms.	SC09-2948 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	Provides general information about APPC, CPI-C, and SNA sense codes that you may encounter when using DB2 Universal Database products. Available in HTML format only.	No form number db2apx70	db2ap
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or Java (JDBC and SQLJ). Discussion topics include writing stored procedures, writing user-defined functions, creating user-defined types, using triggers, and developing applications in partitioned environments or with federated systems.	SC09-2949 db2a0x70	db2a0
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	SC09-2950 db2l0x70	db2l0
Command Reference	Explains how to use the Command Line Processor and describes the DB2 commands that you can use to manage your database.	SC09-2951 db2n0x70	db2n0
Connectivity Supplement	Provides setup and reference information on how to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA application requesters with DB2 Universal Database servers. This book also details how to use DRDA application servers with DB2 Connect application requesters. Available in HTML and PDF only.	No form number db2h1x70	db2h1

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Data Movement Utilities Guide and Reference	Explains how to use DB2 utilities, such as import, export, load, AutoLoader, and DPROP, that facilitate the movement of data.	SC09-2955 db2dmx70	db2dm
Data Warehouse Center Administration Guide	Provides information on how to build and maintain a data warehouse using the Data Warehouse Center.	SC26-9993 db2ddx70	db2dd
Data Warehouse Center Application Integration Guide	Provides information to help programmers integrate applications with the Data Warehouse Center and with the Information Catalog Manager.	SC26-9994 db2adx70	db2ad
DB2 Connect User's Guide	Provides concepts, programming, and general usage information for the DB2 Connect products.	SC09-2954 db2c0x70	db2c0
DB2 Query Patroller Administration Guide	Provides an operational overview of the DB2 Query Patroller system, specific operational and administrative information, and task information for the administrative graphical user interface utilities.	SC09-2958 db2dwx70	db2dw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patroller.	SC09-2960 db2wwx70	db2ww
Glossary	Provides definitions for terms used in DB2 and its components. Available in HTML format and in the *SQL Reference*.	No form number db2t0x70	db2t0
Image, Audio, and Video Extenders Administration and Programming	Provides general information about DB2 extenders, and information on the administration and configuration of the image, audio, and video (IAV) extenders and on programming using the IAV extenders. It includes reference information, diagnostic information (with messages), and samples.	SC26-9929 dmbu7x70	dmbu7
Information Catalog Manager Administration Guide	Provides guidance on managing information catalogs.	SC26-9995 db2dix70	db2di

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Information Catalog Manager Programming Guide and Reference	Provides definitions for the architected interfaces for the Information Catalog Manager.	SC26-9997 db2bix70	db2bi
Information Catalog Manager User's Guide	Provides information on using the Information Catalog Manager user interface.	SC26-9996 db2aix70	db2ai
Installation and Configuration Supplement	Guides you through the planning, installation, and setup of platform-specific DB2 clients. This supplement also contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, the configuration of distributed requests, and accessing heterogeneous data sources.	GC09-2957 db2iyx70	db2iy
Message Reference	Lists messages and codes issued by DB2, the Information Catalog Manager, and the Data Warehouse Center, and describes the actions you should take. You can order both volumes of the Message Reference in the English language in North America with the form number SBOF-8932.	Volume 1 SC09-2978 db2m1x70 Volume 2 SC09-2979 db2m2x70	db2m0
OLAP Integration Server Administration Guide	Explains how to use the Administration Manager component of the OLAP Integration Server.	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	Explains how to create and populate OLAP metaoutlines using the standard OLAP Metaoutline interface (not by using the Metaoutline Assistant).	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	Explains how to create OLAP models using the standard OLAP Model Interface (not by using the Model Assistant).	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	Provides configuration and setup information for the OLAP Starter Kit.	SC27-0702 db2ipx70	db2ip
OLAP Spreadsheet Add-in User's Guide for Excel	Describes how to use the Excel spreadsheet program to analyze OLAP data.	SC27-0786 db2epx70	db2ep

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	Describes how to use the Lotus 1-2-3 spreadsheet program to analyze OLAP data.	SC27-0785 db2tpx70	db2tp
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9920 db2e0x70	db2e0
Spatial Extender User's Guide and Reference	Provides information about installing, configuring, administering, programming, and troubleshooting the Spatial Extender. Also provides significant descriptions of spatial data concepts and provides reference information (messages and SQL) specific to the Spatial Extender.	SC27-0701 db2sbx70	db2sb
SQL Getting Started	Introduces SQL concepts and provides examples for many constructs and tasks.	SC09-2973 db2y0x70	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. This book also includes information about release-to-release incompatibilities, product limits, and catalog views. You can order both volumes of the *SQL Reference* in the English language in North America with the form number SBOF-8933.	Volume 1 SC09-2974 db2s1x70 Volume 2 SC09-2975 db2s2x70	db2s0
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. This book explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2956 db2f0x70	db2f0
Text Extender Administration and Programming	Provides general information about DB2 extenders and information on the administration and configuring of the text extender and on programming using the text extenders. It includes reference information, diagnostic information (with messages) and samples.	SC26-9930 desu9x70	desu9

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	GC09-2850 db2p0x70	db2p0
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 7.	SC09-2976 db2q0x70	db2q0
DB2 Installation and Configuration Information			
DB2 Connect Enterprise Edition for OS/2 and Windows Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2953 db2c6x70	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Enterprise Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2952 db2cyx70	db2cy
DB2 Connect Personal Edition Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Personal Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for all supported clients.	GC09-2967 db2c1x70	db2c1
DB2 Connect Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Connect Personal Edition on all supported Linux distributions.	GC09-2962 db2c4x70	db2c4
DB2 Data Links Manager Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX and Windows 32-bit operating systems.	GC09-2966 db2z6x70	db2z6

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2964 db2v3x70	db2v3
DB2 Enterprise - Extended Edition for Windows Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2963 db2v6x70	db2v6
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. This book also contains installation and setup information for many supported clients.	GC09-2968 db2i2x70	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2970 db2ixx70	db2ix
DB2 for Windows Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2971 db2i6x70	db2i6
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2 and Windows 32-bit operating systems.	GC09-2969 db2i1x70	db2i1
DB2 Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on all supported Linux distributions.	GC09-2972 db2i4x70	db2i4

Table 36. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Query Patroller Installation Guide	Provides installation information about DB2 Query Patroller.	GC09-2959 db2iwx70	db2iw
DB2 Warehouse Manager Installation Guide	Provides installation information for warehouse agents, warehouse transformers, and the Information Catalog Manager.	GC26-9998 db2idx70	db2id
Cross-Platform Sample Programs in HTML			
Sample programs in HTML	Provides the sample programs in HTML format for the programming languages on all platforms supported by DB2. The sample programs are provided for informational purposes only. Not all samples are available in all programming languages. The HTML samples are only available when the DB2 Application Development Client is installed. For more information on the programs, refer to the *Application Building Guide*.	No form number	db2hs
Release Notes			
DB2 Connect Release Notes	Provides late-breaking information that could not be included in the DB2 Connect books.	See note #2.	db2cr
DB2 Installation Notes	Provides late-breaking installation-specific information that could not be included in the DB2 books.	Available on product CD-ROM only.	
DB2 Release Notes	Provides late-breaking information about all DB2 products and features that could not be included in the DB2 books.	See note #2.	db2ir

Notes:

1. The character *x* in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d0e70` identifies the English version of the *Administration Guide* and the file name `db2d0f70` identifies the French version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	Identifier
Brazilian Portuguese	b

Bulgarian	u
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Russian	r
Simp. Chinese	c
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

2. Late breaking information that could not be included in the DB2 books is available in the Release Notes in HTML format and as an ASCII file. The HTML version is available from the Information Center and on the product CD-ROMs. To view the ASCII file:
   - On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
     - `/usr/lpp/db2_07_01` on AIX
     - `/opt/IBMdb2/V7.1` on HP-UX, PTX, Solaris, and Silicon Graphics IRIX
     - `/usr/IBMdb2/V7.1` on Linux.
   - On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed. On OS/2 platforms, you can also double-click the **IBM DB2** folder and then double-click the **Release Notes** icon.

Printing the PDF Books

If you prefer to have printed copies of the books, you can print the PDF files found on the DB2 publications CD-ROM. Using the Adobe Acrobat Reader, you can print either the entire book or a specific range of pages. For the file name of each book in the library, see Table 36 on page 482.

You can obtain the latest version of the Adobe Acrobat Reader from the Adobe Web site at http://www.adobe.com.

The PDF files are included on the DB2 publications CD-ROM with a file extension of PDF. To access the PDF files:

1. Insert the DB2 publications CD-ROM. On UNIX-based platforms, mount the DB2 publications CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
2. Start the Acrobat Reader.
3. Open the desired PDF file from one of the following locations:
   - On OS/2 and Windows platforms:

     x:\doc*language* directory, where *x* represents the CD-ROM drive and *language* represent the two-character country code that represents your language (for example, EN for English).
   - On UNIX-based platforms:

     /*cdrom*/doc/*%L* directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

You can also copy the PDF files from the CD-ROM to a local or network drive and read them from there.

Ordering the Printed Books

You can order the printed DB2 books either individually or as a set (in North America only) by using a sold bill of forms (SBOF) number. To order books, contact your IBM authorized dealer or marketing representative, or phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada. You can also order the books from the Publications Web page at http://www.elink.ibmlink.ibm.com/pbl/pbl.

Two sets of books are available. SBOF-8935 provides reference and usage information for the DB2 Warehouse Manager. SBOF-8931 provides reference and usage information for all other DB2 Universal Database products and features. The contents of each SBOF are listed in the following table:

Table 37. Ordering the printed books

SBOF Number	Books Included	
SBOF-8931	• Administration Guide: Planning • Administration Guide: Implementation • Administration Guide: Performance • Administrative API Reference • Application Building Guide • Application Development Guide • CLI Guide and Reference • Command Reference • Data Movement Utilities Guide and Reference • Data Warehouse Center Administration Guide • Data Warehouse Center Application Integration Guide • DB2 Connect User's Guide • Installation and Configuration Supplement • Image, Audio, and Video Extenders Administration and Programming • Message Reference, Volumes 1 and 2	• OLAP Integration Server Administration Guide • OLAP Integration Server Metaoutline User's Guide • OLAP Integration Server Model User's Guide • OLAP Integration Server User's Guide • OLAP Setup and User's Guide • OLAP Spreadsheet Add-in User's Guide for Excel • OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3 • Replication Guide and Reference • Spatial Extender Administration and Programming Guide • SQL Getting Started • SQL Reference, Volumes 1 and 2 • System Monitor Guide and Reference • Text Extender Administration and Programming • Troubleshooting Guide • What's New
SBOF-8935	• Information Catalog Manager Administration Guide • Information Catalog Manager User's Guide • Information Catalog Manager Programming Guide and Reference	• Query Patroller Administration Guide • Query Patroller User's Guide

DB2 Online Documentation

Accessing Online Help

Online help is available with all DB2 components. The following table describes the various types of help.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: `? command` where *command* represents a keyword or the entire command. For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
Client Configuration Assistant Help *Command Center Help* *Control Center Help* *Data Warehouse Center Help* *Event Analyzer Help* *Information Catalog Manager Help* *Satellite Administration Center Help* *Script Center Help*	Explains the tasks you can perform in a window or notebook. The help includes overview and prerequisite information you need to know, and it describes how to use the window or notebook controls.	From a window or notebook, click the **Help** push button or press the **F1** key.

Message Help	Describes the cause of a message and any action you should take.	From the command line processor in interactive mode, enter:
		`? XXXnnnnn`
		where *XXXnnnnn* represents a valid message identifier.
		For example, `? SQL30081` displays help about the SQL30081 message.
		To view message help one screen at a time, enter:
		`? XXXnnnnn | more`
		To save message help in a file, enter:
		`? XXXnnnnn > filename.ext`
		where *filename.ext* represents the file where you want to save the message help.

Type of Help	Contents	How to Access...
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: `help statement` where *statement* represents an SQL statement. For example, `help SELECT` displays help about the SELECT statement. **Note:** SQL help is not available on UNIX-based platforms.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, 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, while `? 08` displays help for the 08 class code.

Viewing Information Online

The books included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can view the online books or sample programs with any browser that conforms to HTML Version 3.2 specifications.

To view online books or sample programs:

- If you are running DB2 administration tools, use the Information Center.
- From a browser, click **File —>Open Page**. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:

    `INSTHOME/sqllib/doc/%L/html/index.htm`

    where *%L* represents the locale name.
  - On other platforms, open the following page:

    `sqllib\doc\html\index.htm`

    The path is located on the drive where DB2 is installed.

If you have not installed the Information Center, you can open the page by double-clicking the **DB2 Information** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Installing the Netscape Browser

If you do not already have a Web browser installed, you can install Netscape from the Netscape CD-ROM found in the product boxes. For detailed instructions on how to install it, perform the following:

1. Insert the Netscape CD-ROM.
2. On UNIX-based platforms only, mount the CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
3. For installation instructions, refer to the `CDNAV`*`nn`*`.txt` file, where *nn* represents your two character language identifier. The file is located at the root directory of the CD-ROM.

Accessing Information with the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on all platforms on which the DB2 administration tools are available.

You can open the Information Center by double-clicking the Information Center icon. Depending on the system you are using, the icon is in the Information folder in the main product folder or the Windows **Start** menu.

You can also access the Information Center by using the toolbar and the **Help** menu on the DB2 Windows platform.

The Information Center provides six types of information. Click the appropriate tab to look at the topics provided for that type.

Tasks Key tasks you can perform using DB2.

Reference DB2 reference information, such as keywords, commands, and APIs.

Books DB2 books.

Troubleshooting
Categories of error messages and their recovery actions.

Sample Programs
Sample programs that come with the DB2 Application Development Client. If you did not install the DB2 Application Development Client, this tab is not displayed.

Web DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides a find feature, so you can look for a specific topic without browsing the lists.

For a full text search, follow the hypertext link in the Information Center to the **Search DB2 Online Information** search form.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server using one of the following methods:

On Windows
Click **Start** and select **Programs —> IBM DB2 —> Information —> Start HTML Search Server**.

On OS/2
Double-click the **DB2 for OS/2** folder, and then double-click the **Start HTML Search Server** icon.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: The Search function is not available in the Linux, PTX, and Silicon Graphics IRIX environments.

Using DB2 Wizards

Wizards help you complete specific administration tasks by taking you through each task one step at a time. Wizards are available through the Control Center and the Client Configuration Assistant. The following table lists the wizards and describes their purpose.

Note: The Create Database, Create Index, Configure Multisite Update, and Performance Configuration wizards are available for the partitioned database environment.

Wizard	Helps You to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click **Add**.
Back up Database	Determine, create, and schedule a backup plan.	From the Control Center, right-click the database you want to back up and select **Backup —> Database Using Wizard**.

Wizard	Helps You to...	How to Access...
Configure Multisite Update	Configure a multisite update, a distributed transaction, or a two-phase commit.	From the Control Center, right-click the **Databases** folder and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, right-click the **Databases** folder and select **Create —> Database Using Wizard**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, right-click the **Tables** icon and select **Create —> Table Using Wizard**.
Create Table Space	Create a new table space.	From the Control Center, right-click the **Table Spaces** icon and select **Create —> Table Space Using Wizard**.
Create Index	Advise which indexes to create and drop for all your queries.	From the Control Center, right-click the **Index** icon and select **Create —> Index Using Wizard**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, right-click the database you want to tune and select **Configure Performance Using Wizard**. For the partitioned database environment, from the Database Partitions view, right-click the first database partition you want to tune and select **Configure Performance Using Wizard**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, right-click the database you want to restore and select **Restore —> Database Using Wizard**.

Setting Up a Document Server

By default, the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, perform the following steps:

1. Copy all files and subdirectories from `\sqllib\doc\html` on your local system to a Web server. Each book has its own subdirectory that contains all the necessary HTML and GIF files that make up the book. Ensure that the directory structure remains the same.

2. Configure the Web server to look for the files in the new location. For information, refer to the NetQuestion Appendix in the *Installation and Configuration Supplement*.
3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. When you are able to view the book files, you can bookmark commonly viewed topics. You will probably want to bookmark the following pages:
   - List of books
   - Tables of contents of frequently used books
   - Frequently referenced articles, such as the ALTER TABLE topic
   - The Search form

For information about how you can serve the DB2 Universal Database online documentation files from a central machine, refer to the NetQuestion Appendix in the *Installation and Configuration Supplement*.

Searching Information Online

To find information in the HTML files, use one of the following methods:

- Click **Search** in the top frame. Use the search form to find a specific topic. This function is not available in the Linux, PTX, or Silicon Graphics IRIX environments.
- Click **Index** in the top frame. Use the index to find a specific topic in the book.
- Display the table of contents or index of the help or the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See "Accessing Information with the Information Center" on page 495 for details.

Appendix K. 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 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 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 which has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information 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 illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

The following terms, which may be denoted by an asterisk(*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries or both and is licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk(**) may be trademarks or service marks of others.

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com
Log on as anonymous. In the directory `/ps/products/db2`, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2
Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

DATA-RCVR-00

Spine information:

IBM® DB2® Universal Database Data Recovery and High Availability Guide and Reference Version 7