

Db2 11.1 for Linux, UNIX, and Windows



# Data Recovery and High Availability Guide and Reference



Db2 11.1 for Linux, UNIX, and Windows



# Data Recovery and High Availability Guide and Reference



---

## Notice regarding this document

This document in PDF form is provided as a courtesy to customers who have requested documentation in this format. It is provided As-Is without warranty or maintenance commitment.



---

# Contents

Notice regarding this document . . . .	iii
--	-----

Figures . . . . .	vii
-------------------	-----

Tables . . . . .	ix
------------------	----

## Chapter 1. High availability . . . . . 1

Outages . . . . .	2
Outage signatures . . . . .	2
Outage cost. . . . .	3
Outage tolerance . . . . .	4
Recovery and avoidance strategies . . . . .	4
High availability strategies. . . . .	5
High availability through redundancy . . . . .	6
High availability through failover . . . . .	6
High availability through clustering . . . . .	7
Database logging . . . . .	23
High availability with Db2 server . . . . .	26
Automatic client reroute roadmap . . . . .	26
Db2 fault monitor facilities for Linux and UNIX . . . . .	35
High availability disaster recovery (HADR). . . . .	39
Db2 High Availability Feature . . . . .	42
High availability through log shipping . . . . .	91
Log mirroring . . . . .	92
High availability through suspended I/O and online split mirror support . . . . .	93
Configuring for high availability . . . . .	106
Configuring TCP/IP keepalive parameters . . . . .	107
Initializing a standby database . . . . .	109
Initializing high availability disaster recovery (HADR) . . . . .	110
Scheduling maintenance for high availability . . . . .	143
Configuring database logging options . . . . .	146
Configuring a clustered environment for high availability . . . . .	161
Synchronizing clocks in a partitioned database environment. . . . .	161
Administering and maintaining a highly available solution . . . . .	163
Log file management . . . . .	163
Minimizing the impact of maintenance on availability . . . . .	175
Synchronizing the primary and standby databases. . . . .	184
HADR delayed replay . . . . .	194
Db2 High availability disaster recovery (HADR) management . . . . .	197
HADR multiple standby databases . . . . .	200
High availability disaster recovery (HADR) in Db2 pureScale environments . . . . .	218
HADR reads on standby feature . . . . .	234
Detecting and responding to system outages in a high availability solution . . . . .	242
Failure management with Db2 cluster services . . . . .	253
Automated cluster caching facility failover . . . . .	253

Automated restart . . . . .	254
Manual intervention in failure situations . . . . .	265

## Chapter 2. Data recovery . . . . . 269

Developing a backup and recovery strategy . . . . .	269
Deciding how often to back up . . . . .	272
Storage considerations for recovery . . . . .	274
Keeping related data together . . . . .	277
Backup and restore operations between different operating systems and hardware platforms . . . . .	277
Log stream merging and log file management in a Db2 pureScale environment . . . . .	278
Log sequence numbers in Db2 pureScale environments . . . . .	283
Recovery history file . . . . .	283
Recovery history file entry status. . . . .	285
Viewing recovery history file entries using the DB_HISTORY administrative view . . . . .	287
Pruning the recovery history file . . . . .	288
Automating recovery history file pruning . . . . .	289
Protecting recovery history file entries from being pruned . . . . .	290
Managing recovery objects . . . . .	291
Deleting database recovery objects using the PRUNE HISTORY command or the db2Prune API. . . . .	292
Automating database recovery object management . . . . .	292
Protecting recovery objects from being deleted . . . . .	294
Managing snapshot backup objects . . . . .	294
Backup image and log file upload to IBM Tivoli Storage Manager (TSM) . . . . .	295
Backup overview . . . . .	300
Backing up data . . . . .	303
Backing up partitioned databases. . . . .	309
Enabling automatic backup. . . . .	311
Backup and restore operations in a Db2 pureScale environment . . . . .	313
Monitoring backup operations. . . . .	318
Optimizing backup performance . . . . .	319
Backup and restore statistics . . . . .	319
Privileges, authorities, and authorization required to use backup . . . . .	321
Compatibility of online backup and other utilities . . . . .	321
Backup examples . . . . .	323
Recover overview . . . . .	324
Recovering data . . . . .	325
Crash recovery . . . . .	341
Disaster recovery . . . . .	356
Version recovery . . . . .	357
Rollforward recovery. . . . .	358
Incremental backup and recovery. . . . .	361
Optimizing recovery performance . . . . .	366

Privileges, authorities, and authorization required to use recover . . . . .	367
Restore overview . . . . .	367
Using restore . . . . .	368
Performing a redirected restore operation . . .	381
Database rebuild . . . . .	389
Monitoring the progress of restore operations	409
Optimizing restore performance . . . . .	409
Privileges, authorities, and authorization required to use restore . . . . .	410
Database schema transporting. . . . .	410
Restore from Db2 pureScale Feature to Db2 Enterprise Server Edition . . . . .	417
Restore from Db2 Enterprise Server Edition to Db2 pureScale instance . . . . .	418
Rollforward overview . . . . .	419
Using rollforward . . . . .	421
Database rollforward operations in a Db2 pureScale environment . . . . .	427
Monitoring a rollforward operation . . . . .	430

Authorization required for rollforward . . . .	431
Rollforward sessions - CLP examples . . . .	432
Data recovery with IBM Tivoli Storage Manager (TSM) . . . . .	436
Configuring a Tivoli Storage Manager client . .	436
Considerations for using Tivoli Storage Manager	439
Db2 Advanced Copy Services (ACS). . . . .	440
Db2 Advanced Copy Services (ACS) best practices . . . . .	441
Restrictions for FlashCopy Limited Function for xLinux and AIX SDK 1.0 . . . . .	441
Enabling Db2 Advanced Copy Services (ACS)	442
Manually installing Tivoli Storage FlashCopy Manager (Linux) . . . . .	446
Db2 Advanced Copy Services (ACS) scripted interface . . . . .	446
Db2 Advanced Copy Services (ACS) API . . .	459

<b>Index . . . . .</b>	<b>503</b>
------------------------	------------



---

## Figures

1. Example of a Failover Clustering configuration	13
2. Active-passive configuration . . . . .	15
3. Mutual takeover configuration . . . . .	16
4. Failover . . . . .	18
5. Circular Logging . . . . .	24
6. Active and archived database logs in rollforward recovery . . . . .	24
7. Synchronization modes for high availability and disaster recovery (HADR) . . . . .	134
8. Reusing log file names . . . . .	165
9. States of the standby database . . . . .	189
10. Database recovery files . . . . .	271
11. Log files in a Db2 pureScale environment	281
12. Creating and updating the recovery history file . . . . .	284
13. Active Database Backups . . . . .	285
14. Inactive Database Backups . . . . .	285
15. Expired Database Backups . . . . .	286
16. Mixed Active, Inactive, and Expired Database Backups . . . . .	287
17. Expired Log Sequence . . . . .	287
18. Log files in a Db2 pureScale environment	316
19. Rolling back units of work (crash recovery)	341
20. Version Recovery . . . . .	357
21. Database Rollforward Recovery . . . . .	359
22. Table Space Rollforward Recovery . . . . .	360
23. Database and table space-level backups of database SAMPLE . . . . .	395
24. Backup log chain for database SAMPLE2	397
25. Backup images available for database SAMPLE . . . . .	398
26. Sets of table spaces and schemas . . . . .	411
27. ORIGINALDB database . . . . .	414
28. TARGETDB database . . . . .	415
29. TARGETDB database after transport . . . . .	416
30. Table space backup requirement . . . . .	426



# Tables

1. Arguments for Creating Containers . . . . .	10	19. Effect of the state of other members in a Db2 pureScale instance on database backup and restore operations . . . . .	314
2. Roadmap to automatic client reroute information . . . . .	26	20. Transport considerations for specific database objects . . . . .	412
3. Supported HADR functionality for different deployments . . . . .	41	21. Comparison of supported features with the FlashCopy Limited Function for xLinux and AIX SDK 1.0 that ships with Db2 with the full version of the IBM Tivoli Storage Manager (TSM) product . . . . .	441
4. Types of quorum device supported by <b>db2haicu</b> . . . . .	49	22. Options written by the library for Db2 ACS . . . . .	447
5. Valid values for the <code>clusterManager</code> attribute . . . . .	59	23. Return codes. . . . .	460
6. Valid values for the <code>quorumDeviceProtocol</code> attribute. . . . .	62	24. Return codes. . . . .	462
7. Valid values for the <code>quorumDeviceName</code> attribute . . . . .	63	25. Return codes. . . . .	464
8. Valid values for the <code>physicalNetworkProtocol</code> attribute. . . . .	64	26. Return codes. . . . .	466
9. How the address space used for HADR communication is determined . . . . .	122	27. Return codes. . . . .	467
10. User Exit Program Return Codes . . . . .	170	28. Return codes. . . . .	469
11. Host name, port number, and instance name for databases. . . . .	210	29. Return codes. . . . .	471
12. Configuration values for each HADR database . . . . .	214	30. Return codes. . . . .	473
13. Configuration values for each HADR database after a role switch . . . . .	215	31. Return codes. . . . .	475
14. Configuration values for each HADR database after a failover . . . . .	216	32. Return codes. . . . .	477
15. Configuration values for a reintegrated standby . . . . .	217	33. Return codes. . . . .	479
16. Configuration values for each HADR database after a failover . . . . .	218	34. Return codes. . . . .	480
17. Endianness of supported Linux and UNIX operating systems Db2 supports . . . . .	278	35. Return codes. . . . .	482
18. Logging-related database configuration parameters . . . . .	279	36. Return codes. . . . .	484
		37. Db2 Advanced Copy Services (ACS) API return codes . . . . .	499
		38. Error codes for script-initiated snapshot operations . . . . .	501



---

## Chapter 1. High availability

The availability of a database solution is a measure of how successful user applications are at performing their required database tasks.

If user applications cannot connect to the database, or if their transactions fail because of errors or time out because of load on the system, the database solution is not very available. If user applications are successfully connecting to the database and performing their work, the database solution is highly available.

Designing a highly available database solution, or increasing the availability of an existing solution requires an understanding of the needs of the applications accessing the database. To get the greatest benefit from the expense of additional storage space, faster processors, or more software licenses, focus on making your database solution as available as required to the most important applications for your business at the time when those applications need it most.

### Unplanned outages

Unexpected system failures that could affect the availability of your database solution to users include: power interruption; network outage; hardware failure; operating system or other software errors; and complete system failure in the event of a disaster. If such a failure occurs at a time when users expect to be able to do work with the database, a highly available database solution must do the following:

- Shield user applications from the failure, so the user applications are not aware of the failure. For example, Db2® Data Server can reroute database client connections to alternate database servers if a database server fails.
- Respond to the failure to contain its effect. For example, if a failure occurs on one machine in a cluster, the cluster manager can remove that machine from the cluster so that no further transactions are routed to be processed on the failed machine.
- Recover from the failure to return the system to normal operations. For example, if standby database takes over database operations for a failed primary database, the failed database might restart, recover, and take over once again as the primary database.

These three tasks must be accomplished with a minimum effect on the availability of the solution to user applications.

### Planned outage

In a highly available database solution, the impact of maintenance activities on the availability of the database to user applications must be minimized as well.

For example, if the database solution serves a traditional store front that is open for business between the hours of 9am to 5pm, then maintenance activities can occur offline, outside of those business hours without affecting the availability of the database for user applications. If the database solution serves an online banking business that is expected to be available for customers to access through the Internet 24 hours per day, then maintenance activities must be run online, or scheduled for off-peak activity periods to have minimal impact on the availability of the database to the customers.

When you are making business decisions and design choices about the availability of your database solution, you must weigh the following two factors:

- The cost to your business of the database being unavailable to customers
- The cost of implementing a certain degree of availability

For example, consider an Internet-based business that makes a certain amount of revenue,  $X$ , every hour the database solution is serving customers. A high availability strategy that saves 10 hours of downtime per year will earn the business  $10X$  extra revenue per year. If the cost of implementing this high availability strategy is less than the expected extra revenue, it would be worth implementing.

---

## Outages

An outage is any disruption in the ability of the database solution to serve user applications. Outages can be classified in two groups: unplanned outages and planned outages.

### Unplanned outages

Examples of unplanned outages include:

- The failure of one component of the system, including hardware or software failure.
- Invalid administrative or user application actions such as accidentally dropping a table that is needed for business-critical transactions.
- Poor performance due to suboptimal configuration, or inadequate hardware or software.

### Planned outages

Examples of planned outages include:

- Maintenance. Some maintenance activities require you to take a complete outage; other maintenance activities can be performed without stopping the database, but can adversely affect performance. The latter is the most common type of planned outage.
- Upgrade. Upgrading your software or hardware can sometimes require a partial or a full outage.

In discussions about availability, the focus is often on disaster scenarios or component failures. However, to design a robust high availability solution, you need to address all of these types of outage.

## Outage signatures

An outage signature is a collection of symptoms and behaviors which characterize an outage. The signature of an outage may vary from temporary performance issues resulting in slow response time for end users to complete site failure.

Consider how these variations impact your business when devising strategies for avoiding, minimizing, and recovering from outages.

### Blackout

A blackout type of outage is experienced when a system is completely unavailable to its end users. This type of outage may be caused by problems at the hardware, operating system, or database level. When a

blackout occurs, it is imperative that the scope of the outage is immediately identified. Is the outage purely at the database level? Is the outage at the instance level? Or is it at the operating system or hardware level?

### **Brownout**

A brownout type of outage is experienced when system performance slows to a point where end users cannot effectively get their work done. The system as a whole may be up and running, but essentially, in the eyes of the end users it is not working as expected. This type of outage may occur during system maintenance windows and peak usage periods. Typically, the CPU and memory are near capacity during such outages. Poorly tuned or overutilized servers often contribute to brownouts.

### **Frequency and duration of outages**

In conversations about database availability, the focus is often on the total amount or the percentage of down time (or conversely the amount of time the database system is available) for a given time period. However, the frequency and duration of planned or unplanned outages makes a significant difference to the impact that those outages have on your business.

Consider a situation in which you have to make some upgrades to your database system that will take seven hours to perform, and you can choose between taking the database system offline for an hour every day during a period of low user activity or taking the database offline for seven hours during the busiest part of your busiest day. Clearly, several small outages would be less costly and harmful to your business activities than the single, seven-hour outage. Now consider a situation in which you have intermittent network failures, possibly for a total of a few minutes every week, which cause a small number of transactions to fail with regular frequency. Those very short outages might end up costing you a great deal of revenue, and irreparably damage the confidence of your customers in your business—resulting in even greater losses of future revenue.

Don't focus exclusively on the total outage (or available) time. Weigh the cost of fewer, longer outages against the cost of multiple, smaller outages when making decisions about maintenance activities or when responding to an unplanned outage. In the middle of an outage, it can be difficult to make such judgments; so create a formula or method to calculate the cost to your business of these outage signatures so that you can make the best choices.

### **Multiple and cascading failures**

When you are designing your database solution to avoid, minimize, and recover from outages, keep in mind the possibility for multiple components to fail at the same time, or even for the failure of one component to cause another component to fail.

## **Outage cost**

The cost of an outage varies from business to business. Each business, as a best practice, should analyze the cost of an outage to their mission critical business processes. The results of this analysis are used to formulate a restoration plan.

This plan includes a priority ordering among restoration activities if more than one process is identified.

## Outage cost

You can estimate the cost to your business of your customer-facing database system being unavailable to process customer transactions. For example, you can calculate an average cost in lost sales revenue for every hour or minute during which that database system is unavailable. Calculating projected losses in revenue as a result of reduced customer confidence is much more difficult, but you should consider this cost when assessing your business's availability requirements.

Consider too the cost of internal database systems being unavailable to your business processes. Something as simple as e-mail or calendar software being unavailable for an hour can cause your business to grind a halt, because employees are unable to do their work.

## Outage tolerance

The tolerance of an outage varies from business to business. Each business, as a best-practice, should analyze the impact of an outage to their mission critical business processes. The results of this analysis are used to formulate a restoration plan.

This plan includes an order of priority to the restoration if more than one process is identified.

## Outage tolerance

A crucial factor in determining your availability needs is to ask how tolerant your business, or a specific system in your business, is to the occurrence of an outage. For example, a restaurant that operates a Web site primarily to publish menu information will not lose much revenue because of an occasional server outage. On the other hand, any outage on a stock exchange server that records transactions would be catastrophic. Thus, using a lot of resources to ensure the availability of the restaurant's server is 99.99% would not be cost-effective, whereas it certainly would be for the stock exchange.

When discussing tolerance two concepts should be kept in mind: time to recovery, and point of recovery.

Time to recovery is the time required to bring a business process or system back online.

Point of recovery is the historical point at which the business process or system is restored. In database terms, a plan would weigh the benefits of a quick restore that loses some transactions versus a complete restore that loses no transactions but which takes longer to perform.

## Recovery and avoidance strategies

When considering purchase and system design choices about availability, it is tempting to dive into long lists of high availability features and technologies. However, best practices with respect to making and keeping your system highly available are just as much about making good design and configuration choices, and designing and practicing sound administrative procedures and emergency plans, as they are about buying technology.



You will get the most comprehensive availability for your investment by first identifying the high availability strategies that best suit your business demands. Then you can implement your strategies, choosing the most appropriate technology.

When designing or configuring your database solution for high availability, consider how outages may be avoided, their impact minimized, and your system quickly recovered.

#### **Avoid outages**

Whenever possible, avoid outages. For example, remove single points of failure to avoid unplanned outages, or investigate methods for performing maintenance activities online to avoid planned outages. Monitor your database system to identify trends in system behavior that indicate problems, and resolve the problems before they cause an outage.

#### **Minimize the impact of outages**

You can design and configure your database solution to minimize the impact of planned and unplanned outages. For example, distribute your database solution so that components and functionality are localized, allowing some user applications to continue processing transactions even when one component is offline.

#### **Recover quickly from unplanned outages**

Make a recovery plan: create clear and well-documented procedures that administrators can follow easily and quickly in the event of an unplanned outage; create clear architectural documents that describe all components of the systems involved; have service agreements and contact information well organized and close to hand. While recovering quickly is vitally important, also know what diagnostic information to collect in order to identify the root cause of the outage and avoid it in the future.

---

## **High availability strategies**

It does not matter to a user why his or her database request failed. Whether a transaction timed out because of bad performance, or a component of the solution failed, or an administrator has taken the database offline to perform maintenance, the result is the same to the user.

The database is unavailable to process requests.

Strategies for improving the availability of your database solution include:

#### **Redundancy**

Having secondary copies of each component of your solution that can take over workload in the event of failure.

#### **System monitoring**

Collecting statistics about the components of your solution to facilitate workload balancing or detecting that components have failed.

#### **Load balancing**

Transferring some workload from an overloaded component of your solution to another component of your solution that has a lighter load.

#### **Failover**

Transferring all workload from a failed component of your solution to a secondary component.

**Maximizing performance**

Reducing the chance that transactions take a very long time to complete or time out.

**Minimizing the impact of maintenance**

Scheduling automated maintenance activities and manual maintenance activities so as to impact user applications as little as possible.

## High availability through redundancy

An important strategy for maintaining high availability is having redundant components. If a component fails, a secondary or backup copy of that component can take over, enabling the database to remain available to user applications.

If a component of the system is not redundant, that component could be a single point of failure for the system.

Redundancy is common in system design:

- Uninterrupted or backup power supplies
- Multiple network fibers between each component
- Bonding or load balancing of network cards
- Multiple hard drives in a redundant array
- Clusters of CPUs

If any one of these components of the system is not redundant, that component could be a single point of failure for the whole system.

You can create redundancy at the database level, by having two databases: a primary database that normally processes all or most of the application workload; and a secondary database that can take over the workload if the primary database fails. In a Db2 High Availability Disaster Recover (HADR) environment, this secondary database is called the standby database.

For Db2 Connect clients, Sysplex workload balancing functionality on Db2 for z/OS® servers provides high availability for client applications that connect directly to a data sharing group. Sysplex workload balancing functionality provides workload balancing and seamless automatic client reroute capability. This support is available for applications that use Java™ clients (JDBC, SQLJ, or pureQuery) or other clients (ODBC, CLI, .NET, OLE DB, PHP, Ruby, or embedded SQL).

## High availability through failover

Failover is the transfer of workload from a primary system to a secondary system in the event of a failure on the primary system. When workload has been transferred like this, the secondary system is said to have taken over the workload of the failed primary system.

**Example 1**

In a clustered environment, if one machine in the cluster fails, cluster managing software can move processes that were running on the machine that failed to another machine in the cluster.

**Example 2**

In a database solution with multiple IBM® Data Servers, if one database becomes unavailable, the database manager can reroute database

applications that were connected to the database server that is no longer available to a secondary database server.

The two most common failover strategies on the market are known as idle standby and mutual takeover:

#### **Idle Standby**

In this configuration, a primary system processes all the workload while a secondary or standby system is idle, or in standby mode, ready to take over the workload if there is a failure on the primary system. In an high availability disaster recovery (HADR) setup, you can have up to three standbys and you can configure each standby to allow read-only workloads.

#### **Mutual Takeover**

In this configuration, there are multiple systems, and each system is the designated secondary for another system. When a system fails, the overall performance is negatively affected because the secondary for the system that failed must continue to process its own workload as well as the workload of the failed system.

## **High availability through clustering**

A cluster is a group of connected machines that work together as a single system. When one machine in a cluster fails, cluster managing software transfers the workload of the failed machine onto other machines.

#### **Heartbeat monitoring**

To detect a failure on one machine in the cluster, failover software can use heartbeat monitoring or keepalive packets between machines to confirm availability. Heartbeat monitoring involves system services that maintain constant communication between all the machines in a cluster. If a heartbeat is not detected, failover to a backup machine starts.

#### **IP address takeover**

When there is a failure on one machine in the cluster, cluster managers can transfer workload from one machine to another by transferring the IP address from one machine to another. This is called IP address takeover, or IP takeover. This transfer is invisible to client applications, which continue to use the original IP address, unaware that the physical machine to which that IP address maps has changed.

The Db2 High Availability Feature enables integration between IBM Db2 server and cluster managing software.

### **Supported cluster management software**

Cluster managing software enables the transfer of Db2 database operations from a failed primary database on one node of the cluster to a secondary database on another node in the cluster.

Db2 database supports the following cluster managing software:

- IBM PowerHA® SystemMirror® for AIX® (formerly known as High Availability Cluster Multi-Processing for AIX or HACMP™)

For detailed information about how to configure PowerHA SystemMirror with Db2 database products, see <http://www.redbooks.ibm.com/abstracts/sg247363.html?Open>.

- Tivoli® System Automation for Multiplatforms.  
For detailed information about how to configure Tivoli System Automation with Db2 database products, see <http://www.redbooks.ibm.com/abstracts/sg247363.html?Open>.
- VERITAS Cluster Server (VCS)  
VCS is a component of the Veritas InfoScale product suite. For detailed information about VCS support matrix, see <https://sort.veritas.com/scfcentral/database>.
- Microsoft Cluster Server, for Windows operating systems  
For detailed information about how to configure Microsoft Cluster Server with Db2 database products, see <http://www.redbooks.ibm.com/abstracts/sg247363.html?Open>.

### **IBM PowerHA SystemMirror for AIX (formerly known as High Availability Cluster Multi-Processing for AIX or HACMP):**

IBM PowerHA SystemMirror for AIX is cluster managing software. The nodes in PowerHA SystemMirror clusters exchange messages called *heartbeats*, or keepalive packets. If a node stops sending these messages, PowerHA SystemMirror invokes failover across the other nodes in the cluster; and when the node that failed is repaired, PowerHA SystemMirror reintegrates it back into the cluster.

There are two types of events: standard events that are anticipated within the operations of PowerHA SystemMirror, 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. This is when a node in the cluster has failed, and PowerHA SystemMirror has initiated failover across the other nodes in the cluster. When planning what should be done as part of the recovery process, PowerHA SystemMirror allows two failover options: hot (or idle) standby, and mutual takeover.

**Note:** When using PowerHA SystemMirror, ensure that Db2 instances are not started at boot time by using the **db2iauto** utility as follows:

```
db2iauto -off InstName
```

where *InstName* is the login name of the instance.

### **Cluster configuration**

In a *hot standby* configuration, the AIX processor node that is the takeover node *is not* running any other workload. In a *mutual takeover* configuration, the AIX processor node that is the takeover node *is* running other workloads.

Generally, in a partitioned database environment, Db2 database runs in mutual takeover mode with database partitions on each node. One exception is a scenario in which the catalog partition is part of a hot standby configuration.

One 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 PowerHA SystemMirror cluster one at a time; it will be faster to start a configuration of multiple clusters rather than one large cluster. PowerHA SystemMirror supports both single and multiple clusters, as long as a node and its backup are in the same cluster.

PowerHA SystemMirror failover recovery allows predefined (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 PowerHA SystemMirror 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 PowerHA SystemMirror. PowerHA SystemMirror can also be used to recover NFS file systems used by Db2 instance main user directories.

Read the PowerHA SystemMirror documentation thoroughly as part of your planning for recovery with Db2 database in a partitioned database environment. You should read the Concepts, Planning, Installation, and Administration guides, then build the recovery architecture for your environment. For each subsystem that you have identified for recovery, based on known points of failure, identify the PowerHA SystemMirror clusters that you need, as well as the recovery nodes (either hot standby or mutual takeover).

If you plan to use PowerHA SystemMirror on two or more computers that share the same home directory, you must install the database manager in the same installation path. Using symbolic links to a similar installation path might cause issues in this scenario. The installation paths must be the same physical path.

It is strongly recommended that both disks and adapters be mirrored in your external disk configuration. For Db2 physical nodes that are configured for PowerHA SystemMirror, care is required to ensure that nodes on the volume group can vary from the shared external disks. In a mutual takeover configuration, this arrangement requires some additional planning, so that the paired nodes can access each other's volume groups without conflicts. In a partitioned database environment, this means that all container names must be unique across all databases for all SMS or DMS table spaces. Automatic storage table spaces manage this requirement for you.

One way to achieve uniqueness is to include the database partition number as part of the name. You can specify a node expression for container string syntax when creating either SMS or DMS containers. When you specify the expression, the node number can be part of the container name or, if you specify additional arguments, the results of those arguments can be part of the container name. Use the argument " \$N" ( blank]\$N) to indicate the node expression. The argument must occur at the end of the container string, and can only be used in one of the following forms:

Table 1. Arguments for Creating Containers. The node number is assumed to be five.

Syntax	Example	Value
blank]\$N	" \$N"	5
blank]\$N+ number]	" \$N+1011"	1016
blank]\$N% number]	" \$N%3"	2
blank]\$N+ number]% number]	" \$N+12%13"	4
blank]\$N% number]+ number]	" \$N%3+20"	22

**Note:**

1. % is modulus.
2. In all cases, the operators are evaluated from left to right.

Following are some examples of how to create containers using this special argument:

- Creating containers for use on a two-node system.

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
(device '/dev/rcont $N' 20000)
```

The following containers would be used:

```
/dev/rcont0 - on Node 0
/dev/rcont1 - on Node 1
```

- Creating containers for use on a four-node system.

```
CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
(file '/DB2/containers/TS2/container $N+100' 10000)
```

The following containers would be used:

```
/DB2/containers/TS2/container100 - on Node 0
/DB2/containers/TS2/container101 - on Node 1
/DB2/containers/TS2/container102 - on Node 2
/DB2/containers/TS2/container103 - on Node 3
```

- Creating containers for use on a two-node system.

```
CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
('/TS3/cont $N%2, '/TS3/cont $N%2+2')
```

The following containers would be used:

```
/TS3/cont0 - on Node 0
/TS3/cont2 - on Node 0
/TS3/cont1 - on Node 1
/TS3/cont3 - on Node 1
```

## Configuring Db2 database partitions for PowerHA SystemMirror

Once configured, each database partition in an instance is started by PowerHA SystemMirror, 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 PowerHA SystemMirror clusters in parallel, than four 16-node clusters.

A script file is packaged with Db2 Enterprise Server Edition to assist in configuring for PowerHA SystemMirror failover or recovery in either hot standby or mutual takeover nodes. The script file is called rc.db2pe.ee for a single node and rc.db2pe.eee for multiple nodes. They are located in the sql/lib/samples/hacmp/es

directory. Copy the appropriate file to /usr/bin on each system in the PowerHA SystemMirror cluster and rename it to rc.db2pe.

In addition, Db2 buffer pool sizes can be customized during failover in mutual takeover configurations from within rc.db2pe. (Buffer pool sizes can be configured to ensure proper resource allocation when two database partitions run on one physical node.)

### **PowerHA SystemMirror event monitoring and user-defined events**

Initiating a failover operation if a process dies on a given node, is an example of a user-defined event. Events must be configured manually as a user defined event as part of the cluster setup.

For detailed information on the implementation and design of highly available IBM Db2 database environments see the IBM Software Library web site (<http://www-01.ibm.com/software/sw-library/type/>).

#### **Related information:**

 [PowerHA SystemMirror Information Center](#)

### **IBM Tivoli System Automation for Multiplatforms (Linux and AIX):**

IBM Tivoli System Automation for Multiplatforms (SA MP) is cluster managing software that facilitates automatic switching of users, applications, and data from one database system to another in a cluster. Tivoli SA MP automates control of IT resources such as processes, file systems, and IP addresses.

Tivoli SA MP provides a framework to automatically manage the availability of what are known as resources. Here are some examples of resources:

- Any piece of software for which start, monitor, and stop scripts can be written to control
- Any network interface card (NIC) to which Tivoli SA MP was granted access. That is, Tivoli SA MP manages the availability of any IP address that a user wants to use by floating that IP address among NICs that it has access to.

For example, both a Db2 instance and high availability disaster recovery have start, stop, and monitor commands. Therefore, Tivoli SA MP scripts can be written to automatically manage these resources. Resources that are closely related (for example, ones that collectively run on the same node at the same time) are called a *resource group*.

#### **Db2 resources**

In a single-partition Db2 environment, a single Db2 instance is running on a server. This Db2 instance has local access to data (its own executable image as well as databases owned by the instance). If this Db2 instance is made accessible to remote clients, an unused IP address must be assigned to this Db2 instance.

The Db2 instance, the local data, and the IP address are all considered resources, which must be automated by Tivoli SA MP. Since these resources are closely related (for example, they collectively run on the same node at the same time), they are called a resource group.

The entire resource group is collocated on one node in the cluster. In the case of a failover, the entire resource group is started on another node.

The following dependencies exist between the resources in the group:

- The Db2 instance must be started after the local disk
- The Db2 instance must be stopped before the local disk
- The HA IP address must be collocated with the instance

### **Disk storage**

The Db2 database can use these resources for local data storage:

- Raw disk (for example, /dev/sda1)
- Logical volume that is managed by Logical Volume Manager (LVM)
- File system (for example, ext3, jfs)

Db2 data can be stored either entirely on one or more raw disks, entirely on logical volumes, entirely on file systems, or on a mixture of all three. Any executables need to be on a file system of some sort.

### **Db2 database requirements for the HA IP address**

The Db2 database has no special requirements for the IP address. It is not necessary to define a highly available IP address in order for the instance to be considered highly available. However, it is important to remember that the IP address that is protected (if any) is the client's access point to the data, and as such, this address must be known by all clients. In practice, it is recommended that this IP address be the one that is used by the clients in their **CATALOG TCPIP NODE** commands.

### **Tivoli SA MP resource groups**

IBM Tivoli System Automation for Multiplatforms is a product that provides high availability by automating resources such as processes, applications, IP addresses, and others in Linux-based clusters. To automate an IT resource (such as an IP address), the resource must be defined to Tivoli SA MP. Furthermore, these resources must all be contained in at least one resource group. If these resources are always required to be hosted on the same machine, they should all be placed in the same resource group.

To be managed and automated with Tivoli SA MP, every application needs to be defined as a resource. Application resources are usually defined in the generic resource class IBM.Application. In this resource class, there are several attributes that define a resource, but at least three of them are application-specific:

- StartCommand
- StopCommand
- MonitorCommand

These commands can be scripts or binary executables.

### **Setting up Tivoli SA MP with your Db2 environment**

For detailed configuration information to help you set up SA MP to work with your Db2 environment, search the IBM Tivoli System Automation for Multiplatforms (SA MP) section of Tivoli documentation central here.



## Microsoft Failover Clustering support (Windows):

Microsoft Failover Clustering supports clusters of servers on Windows operating systems. It automatically detects and responds to server or application failure, and can balance server workloads.

### Introduction

Microsoft Failover Clustering is a feature of Windows server operating systems. It is the software that supports the connection of two servers (up to four servers in Datacenter Server) into a cluster for high availability and easier management of data and applications. Failover Clustering can also automatically detect and recover from server or application failures. It can be used to move server workloads to balance machine utilization and to provide for planned maintenance without downtime.

The following Db2 database products have support for Microsoft Failover Clustering:

- Db2 Connect server products (Db2 Connect Enterprise Edition, Db2 Connect Application Server Edition, Db2 Connect Unlimited Edition for System i® and Db2 Connect Unlimited Edition for System z®).
- Db2 Advanced Enterprise Server Edition
- Db2 Enterprise Server Edition
- Db2 Workgroup Server Edition

### Db2 Failover Clustering components

A cluster is a configuration of two or more nodes, each of which is an independent computer system. The cluster appears to network clients as a single server.

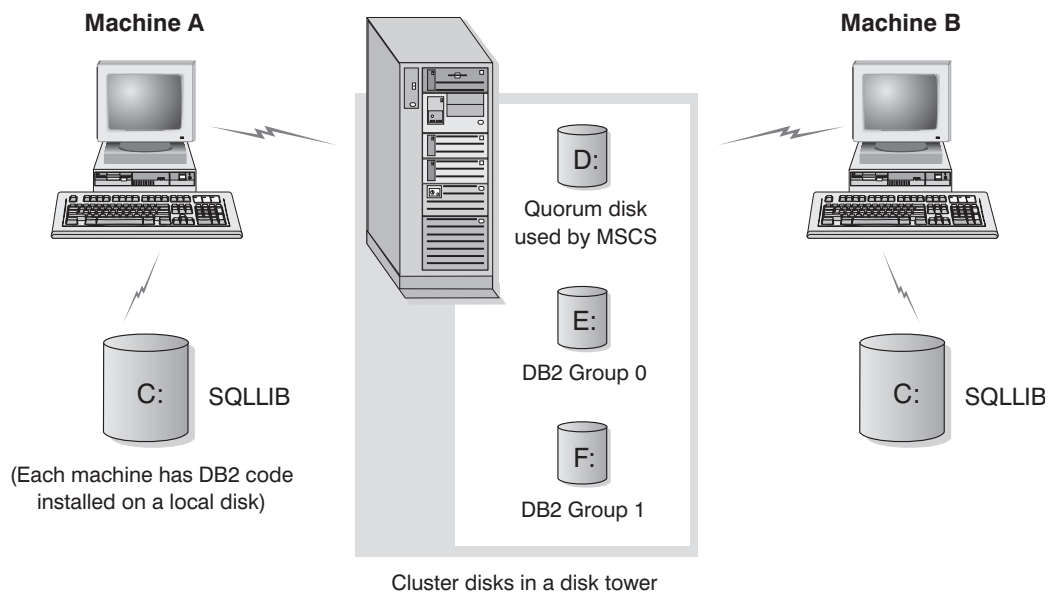


Figure 1. Example of a Failover Clustering configuration

The nodes in a Failover Clustering cluster are connected by one or more shared storage buses and one or more physically independent networks. The network that connects only the servers but does not connect the clients to the cluster is referred to as a *private* network. The network that supports client connections is referred to

as the *public* network. There are one or more local disks on each node. Each shared storage bus attaches to one or more disks. Each disk on the shared bus is owned by only one node of the cluster at a time. The Db2 software resides on the local disk. Db2 database files (for example tables, indexes, log files) reside on the shared disks. Because Microsoft Failover Clustering does not support the use of raw partitions in a cluster, it is not possible to configure Db2 to use raw devices in a Microsoft Failover Clustering environment.

### The Db2 resource

In a Microsoft Failover Clustering environment, a resource is an entity that is managed by the clustering software. For example, a disk, an IP address, or a generic service can be managed as a resource. Db2 integrates with Microsoft Failover Clustering by creating its own resource type called Db2 Server. Each Db2 Server resource manages a Db2 instance, and in a partitioned database environment, each Db2 Server resource manages a database partition. The name of the Db2 Server resource is the instance name, although in the case of a partitioned database environment, the name of the Db2 Server resource consists of both the instance name and the database partition (or node) number.

### Pre-online and post-online scripts

You can run scripts both before and after a Db2 resource is brought online. These scripts are referred to as *pre-online* and *post-online* scripts. Pre-online and post-online scripts are .BAT files that can run Db2 and system commands.

In a situation when multiple instances of Db2 might be running on the same machine, you can use the pre-online and post-online scripts to adjust the configuration so that both instances can be started successfully. In the event of a failover, you can use the post-online script to perform manual database recovery. Post-online script can also be used to start any applications or services that depend on Db2.

### The Db2 group

Related or dependent resources are organized into resource groups. All resources in a group move between cluster nodes as a unit. For example, in a typical Db2 single partition cluster environment, there is a Db2 group that contains the following resources:

1. Db2 resource. The Db2 resource manages the Db2 instance (or node).
2. IP Address resource. The IP Address resource allows client applications to connect to the Db2 server.
3. Network Name resource. The Network Name resource allows client applications to connect to the Db2 server by using a name rather than using an IP address. The Network Name resource has a dependency on the IP Address resource. The Network Name resource is optional. (Configuring a Network Name resource can affect the failover performance.)
4. One or more Physical Disk resources. Each Physical Disk resource manages a shared disk in the cluster.

**Note:** When you use MSCS, it is recommended that you use drive letters when you determine the path for Db2 databases and table space containers. If you are unable to use drive letters, or if the **db2mscs** utility is unable to obtain the drive letter of the disk resource, you must use INSTPROF\_PATH in the db2mscs.cfg file to specify a path on MSCS disks.

**Note:** The Db2 resource is configured to depend on all other resources in the same group so the Db2 server can be started only after all other resources are online.

### Failover configurations

Two types of configuration are available:

- Active-passive
- Mutual takeover

In a partitioned database environment, the clusters do not all need to have the same type of configuration. You can have some clusters that are set up to use active-passive, 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 passive standby configuration, and one machine that is not configured for failover support.

#### Active-passive configuration

In an active-passive configuration, one machine in the Microsoft Failover Clustering cluster provides dedicated failover support, and the other machine participates in the database system. If the machine for the database system fails, the database server on it is started on the failover machine. If, in a partitioned database environment, you are running multiple logical nodes on a machine and it fails, the logical nodes are started on the failover machine. Figure 2 shows an example of an active-passive configuration.

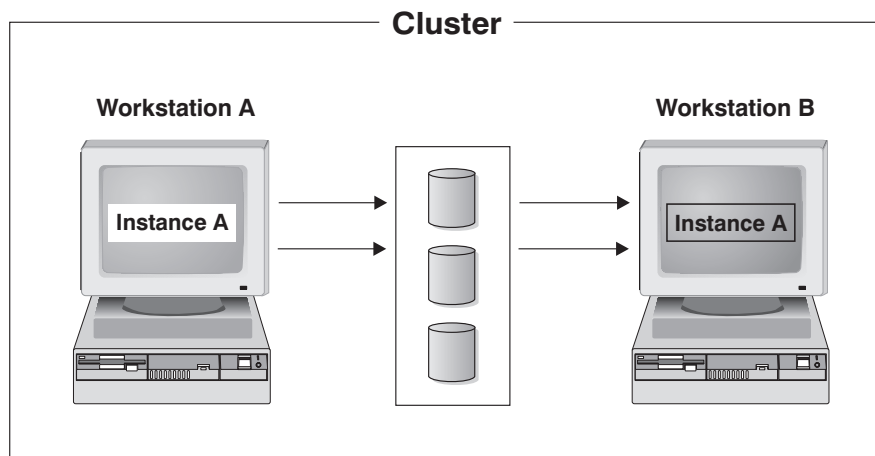


Figure 2. Active-passive configuration

#### 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 that is running on it). If one of the workstations in the Microsoft Failover Clustering cluster fails, the database server on the failing machine is 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 3 on page 16 shows an example of a mutual takeover configuration.

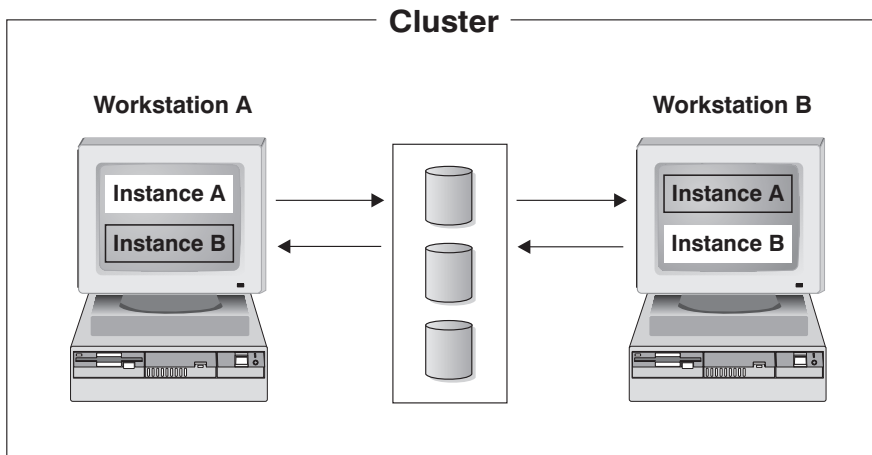


Figure 3. Mutual takeover configuration

### Windows Server 2008 Failover Clustering support

To configure partitioned Db2 database systems to run on Windows Server 2008<sup>1</sup> failover clusters:

1. Follow the same procedures as described in the white paper “Implementing IBM Db2 9.7 Enterprise Server edition with Microsoft Failover Clustering”,<sup>2</sup> which is available on the developerWorks® website here.
2. Due to changes in the Failover Clustering feature of Windows Server 2008, the following additional setup might be required:
  - In Windows Server 2008 failover clusters, the Windows cluster service is run under a special Local System account, whereas in Windows Server 2003, the Windows cluster service is run under an administrator's account. This affects the operations of the Db2 resource (db2server.dll), which is run under the context of the cluster service account.

If the **DB2\_EXTSECURITY** registry variable is set to YES on a Windows failover cluster, the DB2ADMNS and DB2USERS groups must be domain groups.

When a multiple partition instance is running on a Windows failover cluster, the INSTPROF path must be set to a network path (for example, \\NetName\DB2MSCS-DB2\DB2PROFS). This is done automatically if you use the **db2mscs** command to cluster the Db2 database system.

When the Windows Server 2008 failover cluster is formed, a computer object that represents the new cluster is created in the Active Directory. For example, if the name of the cluster is MYCLUSTER, then a computer object MYCLUSTER is created in the Active Directory. If a user clusters a multiple partition instance and the **DB2\_EXTSECURITY** registry variable is set to YES (the default setting), then this computer object must be added to the DB2ADMNS group. You must do this addition so that the Db2 resource DLL can access the \\NetName\DB2MSCS-DB2\DB2PROFS path. For example, if the Db2 Administrators group is MYDOMAIN\DB2ADMNS, the computer object MYCLUSTER must be added to this group. Lastly, after you add the computer object to the DB2ADMNS group, you must reboot both nodes in the cluster.

1. The procedure that is described in this white paper is not materially different for Windows Server 2012

2. The procedure that is described in this white paper is not materially different for Db2V11.1

- In Windows Server 2008 Failover Clustering, the “cluster fileshare resource” is no longer supported. The cluster file server is used instead. The file share (a regular file share) is based on the cluster file server resource. Microsoft requires that the cluster file servers created in the cluster use Domain Name System (DNS) for name resolution. When you are running multiple partition instances, a file server resource is required to support the file share. The values of the **NETNAME\_NAME**, **NETNAME\_VALUE**, and **NETNAME\_DEPENDENCY** parameters that are specified in the `db2mcs.cfg` file are used to create the file server and file share resources. The *NetName* is based on an IP address, and this *NetName* must be in DNS. For example, if a `db2mcs.cfg` file contains the following parameters, a file share `\\MSCSV\DB2MSCS-DB2` is created:

```
...
NETNAME_NAME  = MSCSN
NETNAME_VALUE = MSCSV
...
```

The name MSCSV must be registered in DNS. Otherwise, the FileServer or the file share that is created for the Db2 cluster fails when DNS resolution is not successful.

### Solaris Operating System cluster support:

Db2 supports a cluster manager available for the Solaris operating system: Veritas Cluster Server (VCS).

**Note:** When using Veritas Cluster Server, ensure that Db2 instances are not started at boot time by using the **db2iauto** utility as follows:

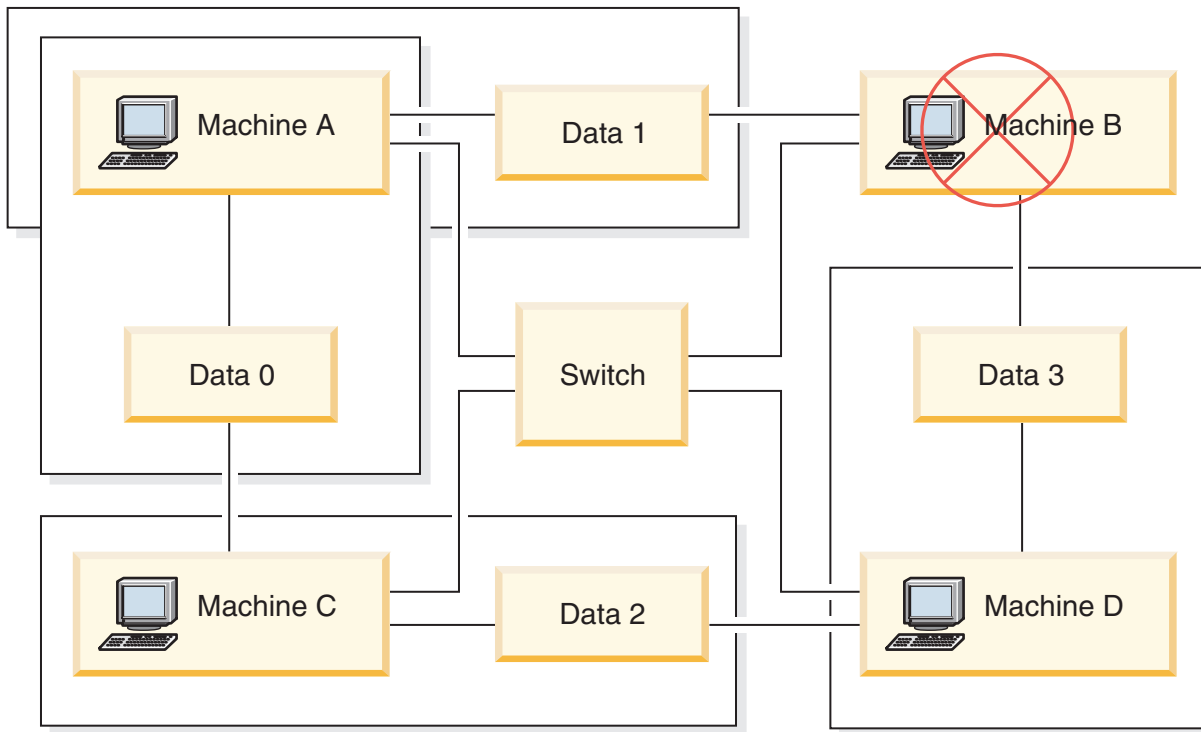
```
db2iauto -off InstName
```

where *InstName* is the login name of the instance.

### High availability

The computer systems that host data services contain many distinct components, and each component has a “mean time before failure” (MTBF) associated with it. The MTBF is the average time that a component will remain usable. The MTBF for a quality hard drive is in the order of one million hours (approximately 114 years). While this seems like a long time, one out of 200 disks is likely to fail within a 6-month period.

Although there are a number of methods to increase availability for a data service, the most common is an HA cluster. A cluster, when used for high availability, consists of two or more machines, a set of private network interfaces, one or more public network interfaces, and some shared disks. This special configuration allows a data service to be moved from one machine to another. By moving the data service to another machine in the cluster, it should be able to continue providing access to its data. Moving a data service from one machine to another is called a *failover*, as illustrated in Figure 4 on page 18.



**Figure 4. Failover.** When Machine B fails its data service is moved to another machine in the cluster so that the data can still be accessed.

The private network interfaces are used to send *heartbeat* messages, as well as control messages, among the machines in the cluster. The public network interfaces are used to communicate directly with clients of the HA cluster. The disks in an HA cluster are connected to two or more machines in the cluster, so that if one machine fails, another machine has access to them.

A data service running on an HA cluster has one or more logical public network interfaces and a set of disks associated with it. The clients of an HA data service connect via TCP/IP to the logical network interfaces of the data service only. If a failover occurs, the data service, along with its logical network interfaces and set of disks, are moved to another machine.

One of the benefits of an HA cluster is that a data service can recover without the aid of support staff, and it can do so at any time. Another benefit is redundancy. All of the parts in the cluster should be redundant, including the machines themselves. The cluster should be able to survive any single point of failure.

Even though highly available data services can be very different in nature, they have some common requirements. Clients of a highly available data service expect the network address and host name of the data service to remain the same, and expect to be able to make requests in the same way, regardless of which machine the data service is on.

Consider a web browser that is accessing a highly available web server. The request is issued with a URL (Uniform Resource Locator), which contains both a host name, and the path to a file on the web server. The browser expects both the host name and the path to remain the same after failover of the web server. If the browser is downloading a file from the web server, and the server is failed over, the browser will need to reissue the request.

Availability of a data service is measured by the amount of time the data service is available to its users. The most common unit of measurement for availability is the percentage of "up time"; this is often referred to as the number of "nines":

99.99% => service is down for (at most) 52.6 minutes / yr  
99.999% => service is down for (at most) 5.26 minutes / yr  
99.9999% => service is down for (at most) 31.5 seconds / yr

When designing and testing an HA cluster:

1. Ensure that the administrator of the cluster is familiar with the system and what should happen when a failover occurs.
2. Ensure that each part of the cluster is truly redundant and can be replaced quickly if it fails.
3. Force a test system to fail in a controlled environment, and make sure that it fails over correctly each time.
4. Keep track of the reasons for each failover. Although this should not happen often, it is important to address any issues that make the cluster unstable. For example, if one piece of the cluster caused a failover five times in one month, find out why and fix it.
5. Ensure that the support staff for the cluster is notified when a failover occurs.
6. Do not overload the cluster. Ensure that the remaining systems can still handle the workload at an acceptable level after a failover.
7. Check failure-prone components (such as disks) often, so that they can be replaced before problems occur.

### **Fault tolerance**

Another way to increase the availability of a data service is fault tolerance. A *fault tolerant* machine has all of its redundancy built in, and should be able to withstand a single failure of any part, including CPU and memory. Fault tolerant machines are most often used in niche markets, and are usually expensive to implement. An HA cluster with machines in different geographical locations has the added advantage of being able to recover from a disaster affecting only a subset of those locations.

An HA cluster is the most common solution to increase availability because it is scalable, easy to use, and relatively inexpensive to implement.

*VERITAS Cluster Server support:*

If you plan to run your Db2 database solution on a Solaris Operating System cluster, you can use VERITAS Cluster Server to manager the cluster.

VERITAS Cluster Server can manage a wide range of applications in heterogeneous environments; and it supports up to 32 node clusters in both storage area network (SAN) and traditional client/server environments.

### **Hardware requirements**

Following is a list of hardware currently supported by VERITAS Cluster Server:

- For server nodes:
  - Any SPARC/Solaris server from Sun Microsystems running Solaris 2.6 or later with a minimum of 128 MB RAM.
- For disk storage:

- EMC Symmetrix, IBM Enterprise Storage Server®, HDS 7700 and 9xxx, Sun T3, Sun A5000, Sun A1000, Sun D1000 and any other disk storage supported by VCS 2.0 or later; your VERITAS representative can confirm which disk subsystems are supported or you can refer to VCS documentation.
- Typical environments will require mirrored private disks (in each cluster node) for the Db2 binaries and shared disks between nodes for the Db2 data.
- For network interconnects:
  - For the public network connections, any network connection supporting IP-based addressing.
  - For the heartbeat connections (internal to the cluster), redundant heartbeat connections are required; this requirement can be met through the use of two additional Ethernet controllers per server or one additional Ethernet controller per server and the use of one shared GABdisk per cluster

### Software requirements

The following VERITAS software components are qualified configurations:

- VERITAS Volume Manager 3.2 or later, VERITAS File System 3.4 or later, VERITAS Cluster Server 2.0 or later.
- DB Edition for Db2 for Solaris 1.0 or later.

While VERITAS Cluster Server does not require a volume manager, the use of VERITAS Volume Manager is strongly recommended for ease of installation, configuration and management.

### Failover

VERITAS Cluster Server is an availability clustering solution that manages the availability of application services, such as Db2 database, by enabling application failover. The state of each individual cluster node and its associated software services are regularly monitored. When a failure occurs that disrupts the application service (in this case, the Db2 database service), either VERITAS Cluster Server or the VCS HA-DB2 Agent, or both will detect the failure and automatically take steps to restore the service. The steps take to restore the service can include restarting the Db2 database system on the same node or moving Db2 database system to another node in the cluster and restarting it on that node. If an application needs to be migrated to a new node, VERITAS Cluster Server moves everything associated with the application (that is, network IP addresses, ownership of underlying storage) to the new node so that users will not be aware that the service is actually running on another node. They will still access the service using the same IP addresses, but those addresses will now point to a different cluster node.

When a failover occurs with VERITAS Cluster Server, users might or might not see a disruption in service. This will be based on the type of connection (stateful or stateless) that the client has with the application service. In application environments with stateful connections (like Db2 database), users might see a brief interruption in service and might need to reconnect after the failover has completed. In application environments with stateless connections (like NFS), users might see a brief delay in service but generally will not see a disruption and will not need to log back on.



By supporting an application as a service that can be automatically migrated between cluster nodes, VERITAS Cluster Server can not only reduce unplanned downtime, but can also shorten the duration of outages associated with planned downtime (for maintenance and upgrades). Failovers can also be initiated manually. If a hardware or operating system upgrade must be performed on a particular node, the Db2 database system can be migrated to another node in the cluster, the upgrade can be performed, and then the Db2 database system can be migrated back to the original node.

Applications recommended for use in these types of clustering environments should be crash tolerant. A crash tolerant application can recover from an unexpected crash while still maintaining the integrity of committed data. Crash tolerant applications are sometimes referred to as *cluster friendly* applications. Db2 database system is a crash tolerant application.

### **Shared storage**

When used with the VCS HA-DB2 Agent, Veritas Cluster Server requires shared storage. Shared storage is storage that has a physical connection to multiple nodes in the cluster. Disk devices resident on shared storage can tolerate node failures since a physical path to the disk devices still exists through one or more alternate cluster nodes.

Through the control of VERITAS Cluster Server, cluster nodes can access shared storage through a logical construct called "disk groups". Disk groups represent a collection of logically defined storage devices whose ownership can be atomically migrated between nodes in a cluster. A disk group can only be imported to a single node at any given time. For example, if Disk Group A is imported to Node 1 and Node 1 fails, Disk Group A can be exported from the failed node and imported to a new node in the cluster. VERITAS Cluster Server can simultaneously control multiple disk groups within a single cluster.

In addition to allowing disk group definition, a volume manager can provide for redundant data configurations, using mirroring or RAID 5, on shared storage. VERITAS Cluster Server supports VERITAS Volume Manager and Solstice DiskSuite as logical volume managers. Combining shared storage with disk mirroring and striping can protect against both node failure and individual disk or controller failure.

### **VERITAS Cluster Server Global Atomic Broadcast(GAB) and Low Latency Transport (LLT)**

An internode communication mechanism is required in cluster configurations so that nodes can exchange information concerning hardware and software status, keep track of cluster membership, and keep this information synchronized across all cluster nodes. The Global Atomic Broadcast (GAB) facility, running across a low latency transport (LLT), provides the high speed, low latency mechanism used by VERITAS Cluster Server to do this. GAB is loaded as a kernel module on each cluster node and provides an atomic broadcast mechanism that ensures that all nodes get status update information at the same time.

By leveraging kernel-to-kernel communication capabilities, LLT provides high speed, low latency transport for all information that needs to be exchanged and synchronized between cluster nodes. GAB runs on top of LLT. VERITAS Cluster Server does not use IP as a heartbeat mechanism, but offers two other more reliable options. GAB with LLT, can be

configured to act as a heartbeat mechanism, or a GABdisk can be configured as a disk-based heartbeat. The heartbeat must run over redundant connections. These connections can either be two private Ethernet connections between cluster nodes, or one private Ethernet connection and one GABdisk connection. The use of two GABdisks is not a supported configuration since the exchange of cluster status between nodes requires a private Ethernet connection.

For more information about GAB or LLT, or how to configure them in VERITAS Cluster Server configurations, consult the VERITAS Cluster Server 2.0 User's Guide for Solaris.

### **Bundled and enterprise agents**

An agent is a program that is designed to manage the availability of a particular resource or application. When an agent is started, it obtains the necessary configuration information from VCS and then periodically monitors the resource or application and updates VCS with the status. In general, agents are used to bring resources online, take resources offline, or monitor resources and provide four types of services: start, stop, monitor and clean. Start and stop are used to bring resources online or offline, monitor is used to test a particular resource or application for its status, and clean is used in the recovery process.

A variety of bundled agents are included as part of VERITAS Cluster Server and are installed when VERITAS Cluster Server is installed. The bundled agents are VCS processes that manage predefined resource types commonly found in cluster configurations (that is, IP, mount, process and share), and they help to simplify cluster installation and configuration considerably. There are over 20 bundled agents with VERITAS Cluster Server.

Enterprise agents tend to focus on specific applications such as the Db2 database application. The VCS HA-DB2 Agent can be considered an Enterprise Agent, and it interfaces with VCS through the VCS Agent framework.

### **VCS resources, resource types, and resource groups**

A resource type is an object definition used to define resources within a VCS cluster that will be monitored. A resource type includes the resource type name and a set of properties associated with that resource that are salient from a high availability point of view. A resource inherits the properties and values of its resource type, and resource names must be unique on a cluster-wide basis.

There are two types of resources: persistent and standard (non-persistent). Persistent resources are resources such as network interface controllers (NICs) that are monitored but are not brought online or taken offline by VCS. Standard resources are those whose online and offline status is controlled by VCS.

The lowest level object that is monitored is a resource, and there are various resource types (that is, share, mount). Each resource must be configured into a resource group, and VCS will bring all resources in a particular resource group online and offline together. To bring a resource group online or offline, VCS will invoke the start or stop methods for each of the resources in the group. There are two types of resource groups:

failover and parallel. A highly available Db2 database configuration, regardless of whether it is partitioned database environment or not, will use failover resource groups.

A "primary" or "master" node is a node that can potentially host a resource. A resource group attribute called `systemlist` is used to specify which nodes within a cluster can be primaries for a particular resource group. In a two node cluster, usually both nodes are included in the `systemlist`, but in larger, multi-node clusters that might be hosting several highly available applications there might be a requirement to ensure that certain application services (defined by their resources at the lowest level) can never fail over to certain nodes.

Dependencies can be defined between resource groups, and VERITAS Cluster Server depends on this resource group dependency hierarchy in assessing the impact of various resource failures and in managing recovery. For example, if the resource group ClientApp1 can not be brought online unless the resource group Db2 has already been successfully started, resource group ClientApp1 is considered dependent on resource group Db2.

## Database logging

Database logging is an important part of your highly available database solution design because database logs make it possible to recover from a failure, and they make it possible to synchronize primary and secondary databases.

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.

Two types of database logging are supported: *circular* and *archive*. Each provides a different level of recovery capability:

- "Circular logging"
- "Archive logging" on page 24

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

### Circular logging

Circular logging is the default behavior when a new database is created. (The `logarchmeth1` and `logarchmeth2` database configuration parameters are set to OFF.)

With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken.

As the name suggests, circular logging uses a *ring* of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions. Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version*

recovery.

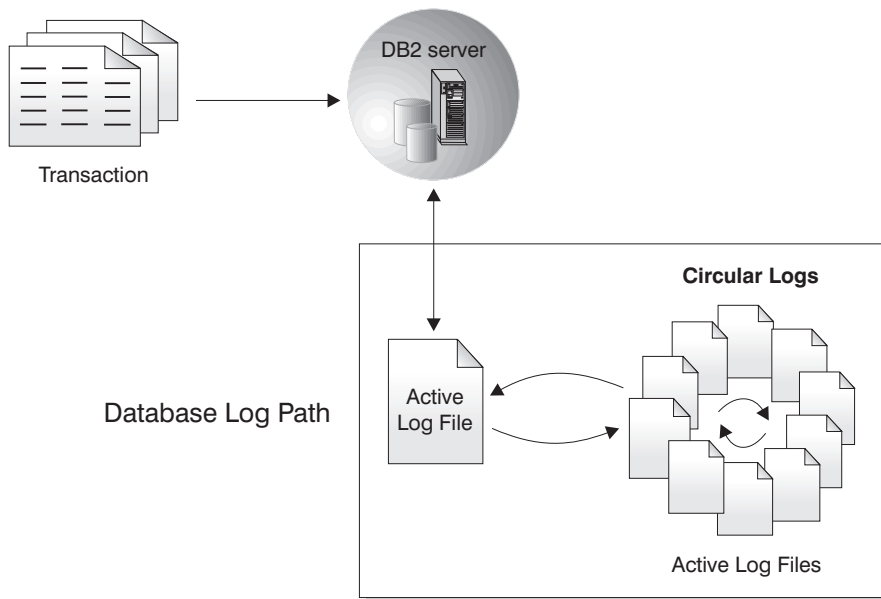


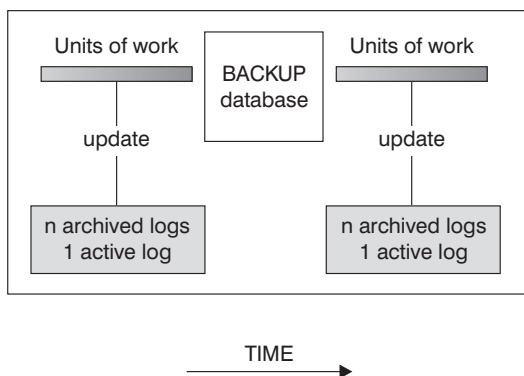
Figure 5. Circular Logging

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

### Archive logging

Archive logging is used specifically for rollforward recovery. Archived logs are log files that are copied from the current log path or from the mirror log path to another location.

You can use the **logarchmeth1** database configuration parameter, the **logarchmeth2** database configuration parameter, or both to allow you or the database manager to manage the log archiving process.



Logs are used between backups to track the changes to the databases.

Figure 6. Active and archived database logs in rollforward recovery. There can be more than one active log in the case of a long-running transaction.

Taking online backups is supported only if you configure the database for archive logging. During an online backup operation, all activities against the database are logged. After an online backup is complete, the database manager forces the currently active log to close, and as a result, it is archived. This process ensures that your online backup has a complete set of archived logs available for recovery. 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. To facilitate this operation, archived logs must be made available when the database is restored.

You can use the **logarchmeth1** and **logarchmeth2** database configuration parameters to specify where archived logs are stored. You can use the **logarchmeth1** parameter to archive log files from the active log path that is set by the **logpath** configuration parameter. You can use the **logarchmeth2** parameter to archive additional copies of log files from the active log path to a second location. If you do not configure mirror logging, the additional copies are taken from the same log path that the **logarchmeth1** parameter uses. If you configure mirror logging, with the **mirrorlogpath** configuration parameter, the **logarchmeth2** configuration parameter archives log files from the mirror log path instead, which can improve resilience during rollforward recovery. The **newlogpath** parameter affects where active logs are stored.

In certain scenarios, you can compress archived log files to help reduce the storage cost that is associated with these files. If the **logarchmeth1** and **logarchmeth2** configuration parameters are set to DISK, TSM, or VENDOR, you can enable archived log file compression by setting the **logarchcompr1** and **logarchcompr2** configuration parameters to ON. If **logarchcompr1** and **logarchcompr2** are set dynamically, any log files that are already archived are not compressed.

If you use the LOGRETAIN option to specify a value that you want to manage the active logs, the database manager renames log files from the active log path after it archives these files and they are no longer needed for crash recovery. If you enable infinite logging, additional space is required for more active log files, so the database server renames the log files after it archives them. The database manager retains up to 8 extra log files in the active log path for renaming purposes.

## Log control files

When a database restarts after a failure, the database manager applies transaction information stored in log files to return the database to a consistent state. To determine which records from the log files need to be applied to the database, the database manager uses information recorded in log control files.

## Redundancy for database resilience

The database manager maintains two copies of the each member's log control file, **SQLLOGCTL.LFH.1** and **SQLLOGCTL.LFH.2**, and two copies of the global log control file, **SQLLOGCTL.GLFH.1** and **SQLLOGCTL.GLFH.2**, so that if one copy is damaged, the database manager can still use the other copy.

## Performance considerations

Applying the transaction information contained in the log control files contributes to the overhead of restarting a database after a failure. You can configure the frequency at which the database manager writes buffer pool pages to disk in order to reduce the number of log records that need to be processed during crash recovery using the “**page\_age\_trgt\_mcr** - Page age target member crash recovery

configuration parameter” in Database Administration Concepts and Configuration Reference.

---

## High availability with Db2 server

IBM Db2 server contains functionality that supports many high availability strategies.

### Automatic client reroute roadmap

Automatic client reroute is an IBM Db2 server feature that redirects client applications from a failed server to an alternate server so the applications can continue their work with minimal interruption. Automatic client reroute can be accomplished only if an alternate server has been specified prior to the failure.

Table 2 lists the relevant topics in each category.

*Table 2. Roadmap to automatic client reroute information*

Category	Related topics
General information	<ul style="list-style-type: none"><li>• “Automatic client reroute limitations” on page 31</li><li>• “Automatic client reroute description and setup”</li><li>• “Automatic client reroute description and setup (Db2 Connect)” in <i>Installing and Configuring Db2 Connect Servers</i></li></ul>
Configuration	<ul style="list-style-type: none"><li>• “Identifying an alternate server for automatic client reroute” on page 30</li><li>• “Configuration of Db2 high availability support for Java clients” in <i>Developing Java Applications</i></li></ul>
Examples	<ul style="list-style-type: none"><li>• “Automatic client reroute examples” on page 33</li></ul>
Interaction with other Db2 features	<ul style="list-style-type: none"><li>• “Configuring automatic client reroute and high availability disaster recovery (HADR)” on page 115</li><li>• “Configuration of Db2 high availability support for Java clients” in <i>Developing Java Applications</i></li></ul>
Troubleshooting	<ul style="list-style-type: none"><li>• “Automatic client reroute configuration for client connection distributor technology” on page 29</li></ul>

**Note:** Automatic client reroute for Db2 for z/OS Sysplex is also available in IBM data server clients and non-Java IBM data server drivers. With this support, applications that access a Db2 for z/OS Sysplex can use automatic client reroute capabilities provided by the client, and are not required to go through a Db2 Connect server. For more information about this feature, see the topic about automatic client reroute (client-side) in the *Db2 Information Center*.

### Automatic client reroute description and setup

The main goal of the automatic client reroute feature is to enable an IBM Data Server Client application to recover from a loss of communications so that the application can continue its work with minimal interruption.

As the name suggests, rerouting is central to the support of continuous operations, but rerouting is only possible when there is an alternate location that is identified to the client connection.

The automatic client reroute feature could be used within the following configurable environments if the server is Db2 :

1. Db2 Enterprise Server Edition with a partitioned database environment
2. Db2 Enterprise Server Edition with the IBM Db2 pureScale® Feature
3. InfoSphere® Replication Server
4. IBM PowerHA SystemMirror for AIX
5. High availability disaster recovery (HADR)
 

Automatic client reroute works in conjunction with HADR and the Db2 pureScale Feature to allow a client application to continue its work with minimal interruption after a failover of the database being accessed.

The seamless automatic client reroute feature is required in the following configuration if the database server is on System i or System z:

1. IBM data server client connects to a z/OS or i5/OS system through a Db2 Connect server which has an alternate server. The automatic client reroute is used between the IBM Data Server Client and two Db2 Connect servers.
2. Db2 Connect clients or server products accessing a Db2 for z/OS Parallel Sysplex® data sharing environment. Automatic client reroute is used between Db2 Connect and the z/OS Parallel Sysplex system. The automatic client reroute feature supports seamless failover between a Db2 Connect-licensed client and the Parallel Sysplex. For more information about seamless failover, see the related links.

In the case of the Db2 Connect server and its alternate, because there is no requirement for the synchronization of local databases, you only need to ensure that both the original and alternate Db2 Connect servers have the target host or System i database cataloged in such a way that it is accessible using an identical database alias.

In order for the Db2 database system to have the ability to recover from a loss of communications, an alternative server location must be specified before the loss of communication occurs. The **UPDATE ALTERNATE SERVER FOR DATABASE** command is used to define the alternate server location on a particular database.

After you have specified the alternate server location on a particular database at the server instance, the alternate server location information is returned to the IBM data server client as part of the connection process. In the case of using automatic client reroute between Db2 Connect clients or server products and a host or System i database server, the remote server must provide one or more alternate addresses for itself. In the case of Db2 for z/OS, multiple addresses are known if the database is a Sysplex data sharing environment. Therefore an alternate server does not need to be cataloged on Db2 Connect. If communication between the client and the server is lost for any reason, the IBM Data Server Client will attempt to reestablish the connection by using the alternate server information. The IBM data server client will attempt to reconnect with a database server which could be the original server, and alternate server listed in the database directory file at the server, or an alternate server that is in the server list returned by the z/OS Parallel Sysplex system. The timing of these attempts to reestablish a connection varies from very rapid attempts initially to a gradual lengthening of the intervals between the attempts.

After a connection is successful, SQL30108N is returned to indicate that a database connection has been reestablished following the communication failure. The hostname or IP address and service name or port number are returned. The IBM

data server client only returns the error for the original communications failure to the application if the reestablishment of the client communications is not possible to either the original or alternative server.

In V10.1 Fix Pack 2 and later fix packs, when connecting to the Db2 for z/OS data sharing group with the workload balance (WLB) feature enabled, non-seamless ACR feature behavior has changed:

- The CLI driver does not immediately look for a new transport upon a connection failure. The CLI driver allocates a transport if the application resubmits the SET statement (special registers) or the SQL statement. When the non-seamless ACR feature is enabled and the WLB feature is disabled, however, the CLI driver immediately looks for a new transport and reconnects to the next available member.
- SQL30108N returns twice to the application if the CLI driver fails to reconnect to members of the primary group and must switch to the alternate group. The error is returned twice when the alternate group is specified in the `db2dsdriver.cfg` file with the **alternategroup** parameter and the **enableAlternateGroupSeamlessAcr** is set to FALSE. The first SQL30108N error with reason code 2 is returned when the existing connection to a member in the current group fails. The second SQL30108N error with reason code 4 is returned when all the connection attempts to all members in the existing primary group fail. The application can then resubmit the SET statement or the SQL statement again for the second time if reconnecting to the alternate group is warranted. The CLI driver tracks the failed member on a same connection handle when the ACR connection error (SQL30108N) is returned to avoid resubmitting the statement to the failed member.

**Note:** SQL30108N is not returned twice in the following scenarios:

- When the Db2 Connect server is used as a gateway.
- When the ACR feature is explicitly enabled without enabling the WLB feature.

When connecting to the Db2 for z/OS data sharing group, the seamless ACR feature and the WLB feature should not be disabled unless directed by IBM support.

Note the following considerations involving alternate server connectivity in a Db2 Connect server environment:

- When using a Db2 Connect server for providing access to a host or System i database on behalf of both remote and local clients, confusion can arise regarding alternate server connectivity information in a system database directory entry. To minimize this confusion, consider cataloging two entries in the system database directory to represent the same host or System i database. Catalog one entry for remote clients and catalog another for local clients.
- Any Parallel Sysplex information that is returned from a target Db2 for z/OS server is kept only in cache at the Db2 Connect server. Only one alternate server is written to disk. When multiple alternates or multiple active servers exist, the information is only maintained in memory and is lost when the process terminates.

Workload balancing and automatic client reroute require the client to have entries for each member in the cluster present in the `/etc/hosts` file. For example:

```
10.10.10.1 hostname01.linux hostname01
10.10.10.2 hostname02.linux hostname02
```



In general, if an alternate server is specified, automatic client reroute is enabled when a communication error is detected. In a high availability disaster recovery (HADR) environment, it is also enabled if SQL1776N is returned back from the HADR standby server.

## **HADR and Db2 pureScale considerations**

When you establish a connection to one member in a Db2 pureScale instance, the server list that is returned contains information not only about all of the members of that instance, but also a hostname and port for the Db2 pureScale instance on the alternate server. If a client cannot connect to one member in the Db2 pureScale instance (or if HADR is configured, to a member on the primary database), it tries another. If it cannot connect to any member, it tries the Db2 pureScale instance at the specified alternate server address (in an HADR environment, the standby database). For better availability, you can use a connection distributor or multi-home DNS entry as alternate server address, but ensure that the distributor or multi-home DNS entry are configured to include multiple members of the alternate server.

Although it is possible to list only one member for each alternate server, clients cannot access the cluster if the listed member is offline, so it is strongly recommended that you define multiple members from the cluster using alternate group method. See the “Related links” for information on how to do this.

Other reroute options include:

### **Client affinity**

List the primary and standby members so that the client tries both. Client affinity and ACR cannot be used together. The alternate server specifies by ACR is ignored when client affinity is enabled. Alternate groups cannot be defined when client affinity is enabled. See the “Client affinities for Db2” topic for information.

### **Virtual IP**

Define a virtual IP address that always points to the current primary server.

### **Automatic client reroute configuration for client connection distributor technology:**

Distributor or dispatcher technologies such as WebSphere® Edge Server Load Balancer distribute client application reconnection requests to a defined set of systems if a primary database server fails.

If you are using distributor technology with Db2 automatic client reroute, you must identify the distributor itself as the alternate server to Db2 automatic client reroute.

You might be using distributor technology in an environment similar to the following:

Client -> distributor technology -> (Db2 Connect server 1 or Db2 Connect server 2)  
->Db2 for z/OS

where:

- The distributor technology component has a TCP/IP host name of DThostname
- The Db2 Connect server 1 has a TCP/IP host name of GWYhostname1

- The Db2 Connect server 2 has a TCP/IP host name of GWYhostname2
- The Db2 for z/OS server has a TCP/IP host name of zOShostname

The client is catalogued using **DThostname** in order to utilize the distributor technology to access either of the Db2 Connect servers. The intervening distributor technology makes the decision to use **GWYhostname1** or **GWYhostname2**. Once the decision is made, the client has a direct socket connection to one of these two Db2 Connect gateways. Once the socket connectivity is established to the chosen Db2 Connect server, you have a typical client to Db2 Connect server to Db2 for z/OS connectivity.

For example, assume the distributor chooses **GWYhostname2**. This produces the following environment:

Client -> Db2 Connect server 2 -> Db2 for z/OS

The distributor does not retry any of the connections if there is any communication failure. If you want to enable the automatic client reroute feature for a database in such an environment, the alternative server for the associated database or databases in the Db2 Connect server (Db2 Connect server 1 or Db2 Connect server 2) should be set up to be the distributor (DThostname). Then, if Db2 Connect server 1 locks up for any reason, automatic client rerouting is triggered and a client connection is retried with the distributor as both the primary and the alternate server. This option allows you to combine and maintain the distributor capabilities with the Db2 automatic client reroute feature. Setting the alternate server to a host other than the distributor host name still provides the clients with the automatic client reroute feature. However, the clients will establish direct connections to the defined alternate server and bypass the distributor technology, which eliminates the distributor and the value that it brings.

The automatic client reroute feature intercepts the following SQL codes:

- SQL20157N
- SQL1768N (reason code: 7)

**Note:** Client reroute might not be informed of socket failures in a timely fashion if the setting of the "TCP Keepalive" operating system configurations parameter is too high. (Note that the name of this configuration parameter varies by platform.)

#### **Identifying an alternate server for automatic client reroute:**

Whenever a Db2 server or Db2 Connect server crashes, each client that is connected to that server receives a communications error which terminates the connection resulting in an application error.

In cases where availability is important, implement either a redundant set-up or the ability to fail the server over to a standby node. In either case, the Db2 client code attempts to re-establish the connection to the original server which might be running on a failover node (the IP address fails over as well), or to a new server.

#### **Procedure**

To define a new or alternate server, use the **UPDATE ALTERNATE SERVER FOR DATABASE** or **UPDATE ALTERNATE SERVER FOR LDAP DATABASE** command.

These commands update the alternate server information for a database alias in the system database directory.

### Automatic client reroute limitations:

Consider Db2 database client reroute restrictions when designing your high availability Db2 database solution.

Here is a list of limitations of the Db2 database automatic client reroute feature:

- You cannot use the ACR feature if you enable reads on standby.
- ACR is only supported when the communications protocol used for connecting to the Db2 database server, or to the Db2 Connect Server, is TCP/IP. This means that if the connection is using a different protocol other than TCP/IP, the automatic client reroute feature will not be enabled. Even if the Db2 database is set up for a loopback, TCP/IP communications protocol must be used in order to accommodate the automatic client reroute feature.
- When using automatic reroute between the Db2 Connect clients or server products and a host or System i database server, if you are in the following situations you will have the associated implications:
  - When using a Db2 Connect Server for providing access to a host or System i database on behalf of both remote and local clients, confusion can arise regarding alternate server connectivity information in a system database directory entry. To minimize this confusion, consider cataloging two entries in the system database directory to represent the same host or System i database. Catalog one entry for remote clients and catalog another for local clients.
  - Any SYSPLEX information that is returned from a target Db2 for z/OS server is kept only in cache at the Db2 Connect Server. Only one alternate server is written to disk. When multiple alternates or multiple active servers exist, the information is only maintained in memory and is lost when the process terminates.
- If the connection is reestablished to the alternate server location, any new connection to the same database alias will be connected to the alternate server location. If you want any new connection to be established, to the original location in case the problem on the original location is fixed, there are a couple of options from which to choose:
  - You need to take the alternate server offline and allow the connections to fail back over to the original server. (This assumes that the original server has been cataloged using the **UPDATE ALTERNATE SERVER** command such that it is set to be the alternate location for the alternate server.)
  - You could catalog a new database alias to be used by the new connections.
  - You could uncatalog the database entry and re-catalog it again.
- Db2 supports the automatic client reroute feature for both the client and the server if both the client and server support this feature. Other Db2 database product families do not currently support this feature.
- The behavior of the automatic client reroute feature and the behavior of the automatic client rerouting in a Db2 for z/OS sysplex environment are somewhat different. Specifically:
  - The automatic client reroute feature requires the primary server to designate a single alternative server. This is done using the **UPDATE ALTERNATE SERVER FOR DATABASE** or **UPDATE ALTERNATE SERVER FOR LDAP DATABASE** command issued at the primary server. This command updates the local database directory with the alternate server information so that other applications at the same client have access this information. By contrast, a data-sharing sysplex used for Db2 for z/OS maintains, in memory, a list of one or more servers to which the

client can connect. If a communication failure happens, the client uses that list of servers to determine the location of the appropriate alternative server.

- In the case of the automatic client reroute feature, the server informs the client of the most current special register settings whenever a special register setting is changed. This allows the client, to the best of its ability, to reestablish the runtime environment after a reroute has occurred. By contrast, a Sysplex used for Db2 for z/OS returns the special register settings to the client on commit boundaries therefore any special registers changed within the unit of work that has been rerouted need to be replayed. All others will be replayed automatically.

Full automatic client reroute support is available only between a Linux, UNIX, or Windows client and a Linux, UNIX, or Windows server. It is not available between a Linux, UNIX, or Windows client and a Db2 for z/OS Sysplex server (any supported version); only the reroute capability is supported.

- The Db2 database server installed in the alternate host server must be the same version (but could have a higher fix pack) when compared to the Db2 database instance installed on the original host server.
- Regardless of whether you have authority to update the database directory at the client machine, the alternate server information is always kept in memory. In other words, if you did not have authority to update the database directory (or because it is a read-only database directory), other applications will not be able to determine and use the alternate server, because the memory is not shared among applications.
- The same authentication is applied to all alternate locations. This means that the client will be unable to reestablish the database connection if the alternate location has a different authentication type than the original location.
- When there is a communication failure, all session resources such as global temporary tables, identity, sequences, cursors, server options (SET SERVER OPTION) for federated processing and special registers are all lost. The application is responsible to reestablish the session resources in order to continue processing the work. You do not have to run any of the special register statements after the connection is reestablished, because the Db2 database will replay the special register statements that were issued before the communication error. However, some of the special registers will not be replayed. They are:
  - SET ENCRYPTPW
  - SET EVENT MONITOR STATE
  - SET SESSION AUTHORIZATION
  - SET TRANSFORM GROUP

When you have a problem with Db2 Connect, you should refer to the list of restricted special registers specific to the Db2 Connect product on a data server.

- If, after the connection is reestablished following a communication failure and the client is using CLI, JCC Type 2 or Type 4 drivers, then those SQL and XQuery statements that have been prepared against the original server are implicitly re-prepared with the new server. However, embedded SQL routines (for example, SQC or SQX applications), are not re-prepared with the new server.
- Do not run high availability disaster recovery (HADR) commands (**START HADR**, **STOP HADR**, or **TAKEOVER HADR**) on client reroute-enabled database aliases. HADR commands are implemented to identify the target database using database aliases. Consequently, if the target database has an alternative database defined, it is difficult for HADR commands to determine the database on which the command is actually operating. A client might need to connect using a client reroute-enabled alias, but HADR commands must be applied on a specific

database. To accommodate this, you can define aliases specific to the primary and standby databases and only run HADR commands on those aliases.

- Because each database server can only have one alternate server defined, if you have a HADR multiple standby setup, you need to select one standby database (likely the principal standby) as the alternate server of the primary.

An alternate way to implement automatic client rerouting is to use the DNS entry to specify an alternate IP address for a DNS entry. The idea is to specify a second IP address (an alternate server location) in the DNS entry; the client would not know about an alternate server, but at connect time Db2 database system would alternate between the IP addresses for the DNS entry.

### **Automatic client reroute examples:**

Automatic client reroute (ACR) can reroute client applications away from a failed database server to a secondary database server previously identified and configured for this purpose. You can easily create client applications that test and demonstrate this Db2 server functionality.

Here is an automatic client reroute example for a client application (shown using pseudo-code only):

```
int checkpoint = 0;

check_sqlca(unsigned char *str, struct sqlca *sqlca)
{
    if (sqlca->sqlcode == -30081)
    {
        // as communication is lost, terminate the application right away
        exit(1);
    }
    else
    {
        // print out the error
        printf(...);
    }
}

if (sqlca->sqlcode == -30108)
{
    // connection is re-established, re-execute the failed transaction
    if (checkpoint == 0)
    {
        goto checkpt0;
    }
    else if (checkpoint == 1)
    {
        goto checkpt1;
    }
    else if (checkpoint == 2)
    {
        goto checkpt2;
    }
    ....
    exit;
}

main()
{
    connect to mydb;
    check_sqlca("connect failed", &sqlca);

checkpt0:
```

```

EXEC SQL set current schema XXX;
check_sqlca("set current schema XXX failed", &sqlca);

EXEC SQL create table t1...;
check_sqlca("create table t1 failed", &sqlca);

EXEC SQL commit;
check_sqlca("commit failed", &sqlca);

if (sqlca.sqlcode == 0)
{
    checkpoint = 1;
}

ckpt1:
EXEC SQL set current schema YYY;
check_sqlca("set current schema YYY failed", &sqlca);

EXEC SQL create table t2...;
check_sqlca("create table t2 failed", &sqlca);

EXEC SQL commit;
check_sqlca("commit failed", &sqlca);

if (sqlca.sqlcode == 0)
{
    checkpoint = 2;
}

...
}

```

At the client machine, the database called “mydb” is cataloged which references a node “hornet” where “hornet” is also cataloged in the node directory (hostname “hornet” with port number 456).

### Example involving a non-HADR database

At the server “hornet” (hostname equals hornet with a port number), a database “mydb” is created. Furthermore, the database “mydb” is also created at the alternate server (hostname “montero” with port number 456). The alternate server for database “mydb” at server “hornet” needs to be updated as follows:

```
db2 update alternate server for database mydb using hostname montero port 456
```

In the sample application above, and without having the automatic client reroute feature set up, if there is a communication error in the create table t1 statement, the application is terminated. With the automatic client reroute feature set up, the Db2 database manager tries to establish the connection to host “hornet” (with port 456) again. If it is still not working, the Db2 database manager tries the alternate server location (host “montero” with port 456). Assuming there is no communication error on the connection to the alternate server location, the application can then continue to run subsequent statements (and to re-run the failed transaction).

### Example involving an HADR database

At the server “hornet” (hostname equals hornet with a port number), primary database “mydb” is created. A standby database is also created at host “montero” with port 456. Information on how to set up HADR for both a primary and standby database is found in “Initializing high availability disaster recovery (HADR)” on page 110. The alternate server for database “mydb” needs to be updated as follows:

```
db2 update alternate server for database mydb using hostname montero port 456
```

In the sample application provided, and without having the automatic client reroute feature set up, if there is a communication error in the create table t1 statement, the application is terminated. With the automatic client reroute feature set up, the Db2 database system tries to establish the connection to host “hornet” (with port 456) again. If it is still not working, the Db2 database system tries the alternate server location (host “montero” with port 456). Assuming there is no communication error on the connection to the alternate server location, the application can then continue to run subsequent statements (and to re-run the failed transaction).

In a Db2 pureScale environment configured with HADR, the behavior is similar. The first time a client connects to the primary database, the server returns the addresses of all members on the primary, plus the alternate server address which specifies the hostname and port of a standby member. If a client cannot connect to one member on the primary, it tries another. If it cannot connect to any member on the primary, it tries the standby at the specified alternate server address.

### Example involving SSL

You can also use client reroute while you are using SSL for your connections too. The setup is the similar to that shown for the previous example for an HADR database.

At the client machine, a database alias “mydb\_ssl” for the database “mydb” is cataloged that references a node, “hornet\_ssl”. “hornet\_ssl” is cataloged in the node directory (hostname is “hornet”, SSL port number is 45678, and the security parameter is set to SSL).

A database alias is also cataloged at the alternate server (hostname is “montero”, SSL port number is 45678, and the security parameter is set to SSL). You also need to update the alternate server for alias “mydb\_ssl” at server “hornet” as shown:

```
db2 update alternate server for database mydb_ssl using hostname montero port 45678
```

In the sample application provided, change the connect statement to connect to mydb\_ssl. Without having the automatic client reroute feature set up, if there is a communication error in the create table t1 statement, the application is terminated. With the automatic client reroute feature set up, the Db2 database manager tries to establish the connection to host “hornet” (with port 45678) using SSL again. If it is still does not work, the Db2 database manager tries the alternate server location (host “montero” at port 45678) using SSL. Assuming there is no communication error on the connection to the alternate server location, the application can then continue to run subsequent statements (and to re-run the failed transaction).

## Db2 fault monitor facilities for Linux and UNIX

Available on UNIX based systems only, Db2 fault monitor facilities keep IBM Db2 server databases up and running by monitoring Db2 database manager instances, and restarting any instance that exits prematurely.

The fault monitor coordinator (FMC) is the process of the fault monitor facility that is started at the UNIX boot sequence. The `init` daemon starts the FMC and will

restart it if it terminates abnormally. The FMC starts one fault monitor for each Db2 instance. Each fault monitor runs as a daemon process and has the same user privileges as the Db2 instance.

Once a fault monitor is started, it will be monitored to make sure it does not exit prematurely. If a fault monitor fails, it will be restarted by the FMC. Each fault monitor will, in turn, be responsible for monitoring one Db2 instance. If the Db2 instance exits prematurely, the fault monitor will restart it. The fault monitor will only become inactive if the **db2stop** command is issued. If a Db2 instance is shut down in any other way, the fault monitor will start it up again.

## Db2 fault monitor restrictions

If you are using a high availability clustering product such as IBM Tivoli System Automation for Multiplatforms (SA MP) or IBM PowerHA SystemMirror for AIX, the fault monitor facility must be turned off since the instance startup and shut down is controlled by the clustering product.

## Differences between the Db2 fault monitor and the Db2 health monitor

The health monitor and the fault monitor are tools that work on a single database instance. The health monitor uses *health indicators* to evaluate the health of specific aspects of database manager performance or database performance. A health indicator measures the health of some aspect of a specific class of database objects, such as a table space. Health indicators can be evaluated against specific criteria to determine the health of that class of database object. In addition, health indicators can generate alerts to notify you when an indicator exceeds a threshold or indicates a database object is in a non-normal state.

By comparison, the fault monitor is solely responsible for keeping the instance it is monitoring up and running. If the Db2 instance it is monitoring terminates unexpectedly, the fault monitor restarts the instance. The fault monitor is not available on Windows.

## Db2 fault monitor registry file

A fault monitor registry file is created for every Db2 database manager instance on each physical machine when the fault monitor daemon is started. The keywords and values in this file specify the behavior of the fault monitors.

The fault monitor registry file can be found in the `/sqllib/` directory and is called `fm.machine_name.reg`. This file can be altered using the **db2fm** command.

If the fault monitor registry file does not exist, the default values will be used.

Here is an example of the contents of the fault monitor registry file:

```
FM_ON = no
FM_ACTIVE = yes
START_TIMEOUT = 600
STOP_TIMEOUT = 600
STATUS_TIMEOUT = 20
STATUS_INTERVAL = 20
RESTART_RETRIES = 3
ACTION_RETRIES = 3
NOTIFY_ADDRESS = instance_name@machine_name
```



## Fault monitor registry file keywords

### **FM\_ON**

Specifies whether or not the fault monitor should be started. If the value is set to NO, the fault monitor daemon will not be started, or will be turned off if it had already been started. The default value is NO.

### **FM\_ACTIVE**

Specifies whether or not the fault monitor is active. The fault monitor will only take action if both **FM\_ON** and **FM\_ACTIVE** are set to YES. If **FM\_ON** is set to YES and **FM\_ACTIVE** is set to NO, the fault monitor daemon will be started, but it will not be active. That means that it will not try to bring Db2 back online if it shuts down. The default value is YES.

### **START\_TIMEOUT**

Specifies the amount of time within which the fault monitor must start the service it is monitoring. The default value is 600 seconds.

### **STOP\_TIMEOUT**

Specifies the amount of time within which the fault monitor must bring down the service it is monitoring. The default value is 600 seconds.

### **STATUS\_TIMEOUT**

Specifies the amount of time within which the fault monitor must get the status of the service it is monitoring. The default value is 20 seconds.

### **STATUS\_INTERVAL**

Specifies the minimum time between two consecutive calls to obtain the status of the service that is being monitored. The default value is 20 seconds.

### **RESTART\_RETRIES**

Specifies the number of times the fault monitor will try to obtain the status of the service being monitored after a failed attempt. Once this number is reached the fault monitor will take action to bring the service back online. The default value is 3.

### **ACTION\_RETRIES**

Specifies the number of times the fault monitor will attempt to bring the service back online. The default value is 3.

### **NOTIFY\_ADDRESS**

Specifies the e-mail address to which the fault monitor will send notification messages. The default is *instance\_name@machine\_name*.

## Configuring Db2 fault monitor using the db2fm command:

You can alter the Db2 fault monitor registry file using the **db2fm** command.

Here are some examples of using the **db2fm** command to update the fault monitor registry file:

### **Example 1: Update START\_TIMEOUT**

To update the START\_TIMEOUT value to 100 seconds for instance DB2INST1, type the following command from a Db2 database command window:

```
db2fm -i db2inst1 -T 100
```

#### Example 2: Update STOP\_TIMEOUT

To update the STOP\_TIMEOUT value to 200 seconds for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -T /200
```

#### Example 3: Update START\_TIMEOUT and STOP\_TIMEOUT

To update the START\_TIMEOUT value to 100 seconds and the STOP\_TIMEOUT value to 200 seconds for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -T 100/200
```

#### Example 4: Turn on fault monitoring

To turn on fault monitoring for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -f yes
```

#### Example 5: Turn off fault monitoring

To turn off fault monitoring for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -f no
```

To confirm that fault monitor is no longer running for DB2INST1, type the following command on UNIX systems:

```
ps -ef|grep -i fm
```

On Linux, type the following command:

```
ps auxw|grep -i fm
```

An entry that shows **db2fmd** and DB2INST1 indicates that the fault monitor is still running on that instance. To turn off the fault monitor, type the following command as the instance owner:

```
db2fm -i db2inst1 -D
```

#### Configuring the Db2 fault monitor using db2fmcu and system commands:

You can configure the Db2 fault monitor using the Db2 fault monitor controller command **db2fmcu** or system commands.

Here are some examples of using **db2fmcu** and system commands to configure the fault monitor:

#### Example 1: Prevent FMC from being launched

You can prevent the fault monitor controller (FMC) from being launched by using the Db2 fault monitor controller command. The **db2fmcu** command must be run as root because it accesses the system's `inittab` file. To block the FMC from being run, type the following command as root:

```
db2fmcu -d
```

**Note:** If you apply a Db2 Data Server fix pack this will be reset so that the `inittab` will again be configured to include the FMC. To prevent the FMC from being launched after you have applied a fix pack, you must reissue the command shown in this example.

### Example 2: Include FMC to be launched

To reverse the **db2fmcu -d** command and reconfigure the `inittab` to include the FMC, type the following command:

```
db2fmcu -u -p fullpath
```

where *fullpath* is the complete path to the **db2fmcd** object, for example `/opt/IBM/db2/bin/db2fmcd`.

### Example 3: Automatically start the Db2 database manager instance

You can also enable FMC to automatically start the instance when the system is first booted. To enable this feature for instance DB2INST1, type the following command:

```
db2iauto -on db2inst1
```

**Note:** On Red Hat Enterprise Linux 6 (RHEL6) systems, the Db2 Fault Monitor Coordinator daemon (`db2fmcd`) does not restart after a system restart, so the Db2 instances will not restart even if it is configured correctly to start automatically. Consult the following technote to enable the fault monitor so that the `db2fmcd` autostarts on RHEL6 systems: <http://www-01.ibm.com/support/docview.wss?uid=swg21497220>.

### Example 4: Disable automatically starting the instance

To turn off the autostart behaviour, type the following command:

```
db2iauto -off db2inst1
```

### Example 5: Prevent fault monitor processes from being launched

You can also prevent fault monitor processes from being launched for a specific instances on the system by changing a field in the global registry record for the instance. To change the global registry field to disable fault monitors for instance DB2INST1, type the following command as root:

```
db2greg -updinstrec instancename=db2inst1!startatboot=0
```

To reverse this command and re-enable fault monitors for instance DB2INST1, type the following command as root:

```
db2greg -updinstrec instancename=db2inst1!startatboot=1
```

## High availability disaster recovery (HADR)

High availability disaster recovery (HADR) provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the *primary database*, to the target databases, called the *standby databases*. HADR supports up to three remote standby servers.

A partial site failure can be caused by a hardware, network, or software (Db2 database system or operating system) failure. Without HADR, a partial site failure requires restarting the database management system (DBMS) server that contains the database. The length of time that it takes to restart the database and the server where it is located is unpredictable. It can take several minutes before the database is brought back to a consistent state and made available. With HADR, a standby database can take over in seconds. Further, you can redirect the clients that used the original primary database to the new primary database by using automatic client reroute or retry logic in the application.

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. However, because HADR uses TCP/IP for communication between the primary and standby databases, they can be situated in different locations. For example, the primary database might be located at your head office in one city, and a standby database might be located at your sales office in another city. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database with full Db2 functionality. After a takeover operation occurs, you can bring the original primary database back up and return it to its primary database status; this is known as *failback*. You can initiate a failback if you can make the old primary database consistent with the new primary database. After you reintegrate the old primary database into the HADR setup as a standby database, you can switch the roles of the databases to enable the original primary database to once again be the primary database.

With HADR, you base the level of protection from potential loss of data on your configuration and topology choices. Some of the key choices that you must make are as follows:

**What level of synchronization will you use?**

Standby databases are synchronized with the primary database through log data that is generated on the primary and shipped to the standbys. The standbys constantly roll forward through the logs. You can choose from four different synchronization modes. In order of most to least protection, these are SYNC, NEARSYNC, ASYNC, and SUPERASYNC.

**Will you use a peer window?**

The peer window feature specifies that the primary and standby databases are to behave as though they are still in peer state for a configured amount of time if the primary loses the HADR connection in peer state. If primary fails in peer or this “disconnected peer” state, the failover to standby occurs with zero data loss. This feature provides the greatest protection.

**How many standbys will you deploy?**

With HADR, you can use up to three standby databases. With multiple standbys, you can achieve both your high availability and disaster recovery objectives with a single technology. For more information, see “HADR multiple standby databases” on page 200.

**Do you want to combine HADR with Db2 pureScale**

The Db2 pureScale feature offers outstanding availability and scalability and can now be combined with HADR to meet both your high availability and disaster recovery needs. For more information, see “High availability disaster recovery (HADR) in Db2 pureScale environments” on page 218.

There are a number of ways that you can use your HADR standby or standbys beyond their HA or DR purpose:

**Reads on standby**

You can use the reads on standby feature to direct read-only workload to one or more standby databases without affecting the HA or DR responsibility of the standby. This feature can help reduce the workload on the primary without affecting the main responsibility of the standby.

Unless you have reads on standby enabled, applications can access the current primary database only. If you have reads on standby enabled,

read-only applications can be redirected to the standby. Applications connecting to the standby database do not affect the availability of the standby in the case of a failover.

### Delayed replay

You can use delayed replay to specify that a standby database is to remain at an earlier point in time than the primary, in terms of log replay. If data is lost or corrupted on the primary, you can recovery this data on the time delayed standby.

### Rolling updates and upgrades

Using an HADR setup, you can make various types of upgrades and Db2 fix pack updates to your databases without an outage. If you are using multiple standby databases, you can perform an upgrade while at the same time keeping the protection provided by HADR. For more information, see “Performing rolling updates in a Db2 high availability disaster recovery (HADR) environment” on page 179.

Table 3 contains an overview of what HADR functionality is supported by each type of HADR setup.

*Table 3. Supported HADR functionality for different deployments*

Functionality or feature	Principal standby	Auxiliary standby	Principal standby in a Db2 pureScale environment
Synchronization mode	All modes are supported	SUPERASYNC mode only	All modes are supported
Reads on standby	Supported	Supported	Not supported
Delayed replay	Supported	Supported	Supported
Log spooling	Supported	Supported	Supported
Tivoli SA MP as cluster manager for automated failover to HADR standby	Supported	Not supported	Not supported
Peer window	Supported	Not supported	Not supported
Network address translation (NAT)	Supported	Supported	Not supported
Automatic client reroute (ACR)	Supported	Supported	Supported
Client affinities (see “Client affinities for clients that connect to Db2” in “Call Level Interface Guide and Reference”).	N/A	N/A	Supported

HADR might be your best option if most or all data in your database requires protection or if you perform DDL operations that must be automatically replicated on a standby database. However, HADR is only one of several replication solutions that are offered in the Db2 product family. The InfoSphere Federation Server software and the Db2 database system include SQL replication and Q replication solutions that you can also use, in some configurations, to provide high availability. These solutions maintain logically consistent copies of database tables at multiple locations. In addition, they provide flexibility and complex functionality

such as support for column and row filtering, data transformation, and updates to any copy of a table. You can also use these solutions in partitioned database environments.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for setting up HADR. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see *Administering databases with task assistants*.

**Related information:**

 [Best practices: High Availability Disaster Recovery](#)

## Db2 High Availability Feature

The Db2 High Availability Feature enables integration between IBM Db2 server and cluster managing software.

When you stop a database manager instance in a clustered environment, you must make your cluster manager aware that the instance is stopped. If the cluster manager is not aware that the instance is stopped, the cluster manager might attempt an operation such as failover on the stopped instance. The Db2 High Availability Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

If the database manager communicates with the cluster manager whenever instance changes require cluster changes, then you are freed from having to perform separate cluster operations after performing instance configuration changes.

The Db2 High Availability Feature is composed of the following elements:

- IBM Tivoli System Automation for Multiplatforms (SA MP) is bundled with Db2 server on AIX and Linux as part of the Db2 High Availability Feature, and integrated with the Db2 installer. You can install, upgrade, or uninstall SA MP using either the Db2 installer or the **installSAM** and **uninstallSAM** scripts that are included in the Db2 server install media.
- In a clustered environment, some database manager instance configuration and administration operations require related cluster configuration changes. The Db2 High Availability Feature (HA) Feature enables the database manager to automatically request cluster manager configuration changes whenever you perform certain database manager instance configuration and administration operations. See: “Configuring a cluster automatically with the Db2 High Availability (HA) Feature” on page 44
- Db2 high availability instance configuration utility (**db2haicu**) is a text-based utility that you can use to configure and administer your highly available databases in a clustered environment. See: “Db2 high availability instance configuration utility (db2haicu)” on page 53

### Cluster manager integration with the Db2 High Availability Feature

The Db2 High Availability Feature enables integration between IBM Db2 server and cluster managing software.

When you stop a database manager instance in a clustered environment, you must make your cluster manager aware that the instance is stopped. If the cluster

manager is not aware that the instance is stopped, the cluster manager might attempt an operation such as failover on the stopped instance. The Db2 High Availability Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

The Db2 High Availability Feature is composed of the following elements:

- IBM Tivoli System Automation for Multiplatforms (SA MP) is bundled with Db2 server on AIX and Linux as part of the Db2 High Availability Feature, and integrated with the Db2 installer. You can install, upgrade, or uninstall SA MP using either the Db2 installer or the **installSAM** and **uninstallSAM** scripts that are included in the Db2 server install media.
- In a clustered environment, some database manager instance configuration and administration operations require related cluster configuration changes. The Db2 High Availability Feature (HA) Feature enables the database manager to automatically request cluster manager configuration changes whenever you perform certain database manager instance configuration and administration operations. See: “Configuring a cluster automatically with the Db2 High Availability (HA) Feature” on page 44
- Db2 high availability instance configuration utility (**db2haicu**) is a text-based utility that you can use to configure and administer your highly available databases in a clustered environment. See: “Db2 high availability instance configuration utility (db2haicu)” on page 53

### **IBM Tivoli System Automation for Multiplatforms (SA MP) base component**

IBM Tivoli System Automation for Multiplatforms (SA MP) provides high availability and disaster recovery capabilities for AIX, Linux, Solaris SPARC, and Windows.

SA MP is integrated with Db2 Enterprise Server Edition, Db2 Advanced Enterprise Server Edition, Db2 Workgroup Server Edition, Db2 Connect Enterprise Edition and Db2 Connect Application Server Edition on AIX, Linux, and Solaris SPARC operating systems.

On Windows operating systems, SA MP is bundled with all of these Db2 database products and features, but it is not integrated with the Db2 database product installer.

You can use this copy of SA MP to manage the high availability of your Db2 database system. You cannot use this copy to manage database systems other than Db2 database systems without buying an upgrade for the SA MP license.

SA MP is the default cluster manager in an IBM Db2 server clustered environment on AIX, Linux, and Solaris SPARC operating systems.

For more information about SA MP, see the IBM Tivoli System Automation for Multiplatforms (SA MP) section of Tivoli documentation central at <https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Tivoli%20Documentation%20Central/page/Tivoli%20System%20Automation%20for%20Multiplatforms>. The list of supported operating systems is also available on the following website: [www.ibm.com/software/tivoli/products/sys-auto-linux/platforms.html](http://www.ibm.com/software/tivoli/products/sys-auto-linux/platforms.html).

## Configuring a cluster automatically with the Db2 High Availability (HA) Feature

In a clustered environment, some database manager instance configuration and administration operations require related cluster configuration changes. The Db2 High Availability Feature (HA) Feature enables the database manager to automatically request cluster manager configuration changes whenever you perform certain database manager instance configuration and administration operations.

### Before you begin

To enable the database manager to perform required cluster configuration for database administration tasks, you must configure the instance for high availability by using **db2haicu** to create a *cluster domain* for the instance. For more information, see: “Configuring a clustered environment using Db2 High Availability Instance Configuration Utility (db2haicu)” on page 45.

### Procedure

When you perform the following database manager instance configuration and administration operations, the database manager automatically performs related cluster manager configuration for you:

- Starting a database using **START DATABASE** or **db2start**.
- Stopping a database using **STOP DATABASE** or **db2stop**.
- Creating a database using **CREATE DATABASE**.
- Adding storage using **CREATE TABLESPACE**.
- Removing storage using **ALTER TABLESPACE DROP** or **DROP TABLESPACE**.
- Adding or removing storage paths using **ALTER DATABASE**.
- Dropping a database using **DROP TABLESPACE**.
- Restoring a database using the **RESTORE DATABASE** or **db2Restore**.
- Specifying the table space containers for redirected restore using **SET TABLESPACE CONTAINERS**.
- Rolling a database forward using **ROLLFORWARD DATABASE** or **db2Rollforward**.
- Recovering a database using **RECOVER DATABASE** or **db2Recover**.
- Creating event monitors using **CREATE EVENT MONITOR**.
- Dropping event monitors using **DROP EVENT MONITOR**.
- Creating and altering external routines using:
  - **CREATE PROCEDURE**
  - **CREATE FUNCTION**
  - **CREATE FUNCTION**
  - **CREATE METHOD**
  - **ALTER PROCEDURE**
  - **ALTER FUNCTION**
  - **ALTER METHOD**
- Dropping external routines using:
  - **DROP PROCEDURE**
  - **DROP FUNCTION**
  - **DROP METHOD**



- Start Db2 High Availability Disaster Recovery (HADR) operations for a database using **START HADR**.
- Stop HADR operations for a database using **STOP HADR**.
- Cause an HADR standby database to take over as an HADR primary database using **TAKEOVER HADR**.
- Setting the database manager configuration parameter **diagpath** or **spm\_log\_path**.
- Setting the database configuration parameter **newlogpath**, **overflowlogpath**, **mirrorlogpath**, or **failarchpath**.
- Dropping a database manager instance using **db2idrop**.

## Results

When the database manager coordinates the cluster configuration changes for database administration tasks listed, you do not have to perform separate cluster manager operations.

## Configuring a clustered environment using Db2 High Availability Instance Configuration Utility (db2haicu)

You can configure and administer your databases in a clustered environment using Db2 high availability instance configuration utility (**db2haicu**). When you specify database manager instance configuration details to **db2haicu**, **db2haicu** communicates the required cluster configuration details to your cluster managing software.

## Before you begin

- There is a set of tasks you must perform before using Db2 high availability instance configuration utility (**db2haicu**). For more information, see: “Db2 High Availability Instance Configuration Utility (db2haicu) prerequisites” on page 86.

## About this task

You can run **db2haicu** interactively, or using an XML input file:

### Interactive mode

When you invoke Db2 high availability instance configuration utility (**db2haicu**) by running the **db2haicu** command without specifying an XML input file with the **-f** parameter, the utility runs in interactive mode. In interactive mode, **db2haicu** displays information and queries you for information in a text-based format. For more information, see: “Running Db2 High Availability Instance Configuration Utility (db2haicu) in interactive mode” on page 56

### Batch mode with an XML input file

You can use the **-f input-file-name** parameter with the **db2haicu** command to run Db2 high availability instance configuration utility (**db2haicu**) with an XML input file specifying your configuration details. Running **db2haicu** with an XML input file is useful when you must perform configuration tasks multiple times, such as when you have multiple database partitions to be configured for high availability. For more information, see: “Running Db2 High Availability Instance Configuration Utility (db2haicu) with an XML input file” on page 56

For a detailed scenario that uses **db2haicu** with both methods to set up an HADR pair, see Automating HADR on Db2 10.1 for Linux, UNIX and Windows Failover Solution Using Tivoli System Automation for Multiplatforms.

Db2 instances that are configured in Version 9.1 for high availability by using the **regdb2salin** script are not supported on Versions 9.5 and later. Use the **db2haicu** utility to configure these instances for high availability.

## Restrictions

There are some restrictions for using the Db2 high availability instance configuration utility (**db2haicu**). For more information, see: “Db2 high availability instance configuration utility (db2haicu) restrictions” on page 89.

## Procedure

Perform the following steps for each database manager instance:

1. Create a new cluster domain.

When you run Db2 high availability instance configuration utility (**db2haicu**) for the first time for a database manager instance, **db2haicu** creates a model of your cluster, called a *cluster domain*. For more information, see: “Creating a cluster domain using Db2 High Availability Instance Configuration Utility (db2haicu)” on page 87.

2. Continue to refine the cluster domain configuration, and administer and maintain the cluster domain

When you are modifying the cluster domain model of your clustered environment using **db2haicu**, the database manager propagates the related changes to your database manager instance and cluster configuration. For more information, see: “Maintaining a cluster domain using Db2 High Availability Instance Configuration Utility (db2haicu)” on page 88.

## What to do next

Db2 high availability instance configuration utility (**db2haicu**) does not have a separate diagnostic log. You can investigate and diagnose **db2haicu** errors using the database manager diagnostic log, **db2diag** log file, and the **db2pd** tool. For more information, see: “Troubleshooting Db2 High Availability Instance Configuration Utility (db2haicu)” on page 89

In addition, the cluster manager that is used by the **db2haicu** utility is IBM Tivoli System Automation for Multiplatforms (SA MP). When troubleshooting cluster manager related issues, it is often necessary to collect trace diagnostic information for SA MP. For this reason, it is highly recommended that you configure trace spooling for SA MP. For more information about configuring trace spooling for SA MP, see <http://www-01.ibm.com/support/docview.wss?uid=swg21375626>. The IBM Tivoli System Automation for Multiplatforms (SA MP) data collection guide also has an appendix that describes trace spooling with different stanzas for the various subcomponents that make up the Reliable Scalable Cluster Technology (RSCT) and SA MP. To review the diagnostic data collection guide for SA MP, see <http://www-01.ibm.com/support/docview.wss?uid=swg21634301>

## Cluster domain:

A cluster domain is a model that contains information about your cluster elements such databases, mount points, and failover policies. You create a cluster domain using Db2 high availability instance configuration utility (**db2haicu**).

**db2haicu** uses the information in the cluster domain to enable configuration and maintenance cluster administration tasks. Also, as part of the Db2 High Availability

(HA) Feature, the database manager uses the information in the cluster domain to perform automated cluster administration tasks.

If you add a cluster element to the cluster domain, then that element will be included in any subsequent **db2haicu** configuration operations, or any automated cluster administration operations that are performed by the database manager as part of the Db2 HA Feature. If you remove a cluster element from the cluster domain, then that element will no longer be included in **db2haicu** operations or database manager automated cluster administration operations. **db2haicu** and the database manager can only coordinate with your cluster manager for cluster elements that are in the cluster domain that you create using **db2haicu**.

You can use **db2haicu** to create and configure the following cluster domain elements:

- Computers or machines (in a cluster domain context, these are referred to as *cluster domain nodes*)
- Network interface cards or NICs (referred to in **db2haicu** as *network interfaces*, *interfaces*, *network adaptors*, or *adaptors*)
- IP addresses
- Databases, including High Availability Disaster Recovery (HADR) primary and standby database pairs
- Database partitions
- Mount points and paths, including those paths that are not critical to failover in the event of a failure
- Failover policies
- Quorum devices

*Cluster management software:*

Cluster management software maximizes the work that a cluster of computers can perform. A cluster manager balances workload to reduce bottlenecks, monitors the health of the elements of the cluster, and manages failover when an element fails.

A cluster manager can also help a system administrator to perform administration tasks on elements in the cluster (by rerouting workload off of a computer that needs to be serviced, for example.)

### Elements of a cluster

To function properly, the cluster manager must be aware of many details related to the elements of the cluster, and the cluster manager must be aware of the relationships between the elements of the cluster.

Here are some examples of cluster elements of which the cluster manager must be aware:

- Physical or virtual computers, machines, or devices in the cluster (in a cluster context, these are referred to as *cluster nodes*)
- Networks that connect the cluster nodes
- Network interfaces cards that connect the cluster nodes to the networks
- IP addresses of cluster nodes
- Virtual or services IP addresses

Here are some examples of relationships of which the cluster manager must be aware:

- Pairs of cluster nodes that have the same software installed and can failover for one another
- Networks that have the same properties and can be used to failover for one another
- The cluster node to which a virtual IP address is currently associated

### **Adding or modifying elements of your cluster**

To make the cluster manager aware of the elements of your cluster and the relationships between those elements, a system administrator must register the elements with the cluster manager. If a system administrator makes a change to the elements of the cluster, the administrator must communicate that change to the cluster manager. Cluster managers have interfaces to help with these tasks.

Cluster administration is challenging because there is an enormous variety of possible cluster elements. An administrator must be an expert in the hardware and operating systems of the cluster nodes, networking protocols and configuration, and the software installed on the cluster nodes such as database software. Registering the elements of the cluster with the cluster management software, or updating the cluster manager after a system change, can be complex and time consuming.

### **Using `db2haicu` to add or modify elements of your cluster**

In a Db2 database solution, you can use the Db2 high availability instance configuration utility (**`db2haicu`**) to register the elements of your cluster with the cluster manager, and to update the cluster manager after making an administrative change to your cluster. Using **`db2haicu`** simplifies these tasks because once you know the model that **`db2haicu`** uses to encapsulate the elements of your cluster and the relationships between those elements, you do not need to be an expert in the idiosyncrasies of your hardware, operating systems, and cluster manager interface to perform the tasks.

#### *Resources and resource groups:*

A *resource* is any cluster element such a cluster node, database, mount point, or network interface card that has been registered with a cluster manager. If an element is not registered with the cluster manager, then the cluster manager will not be aware of that element and the cluster manager will not include that element in cluster managing operations. A *resource group* is a logical collection of resources. The resource group is a very powerful construct because relationships and constraints can be defined on resource groups that simplify performing complex administration tasks on the resources in those groups.

When a cluster manager collects resources into groups, the cluster manager can operate on all those resources collectively. For example, if two databases called database-1 and database-2 belong to the resource group called resource-group-A, then if the cluster manager performs a start operation on resource-group-A then both database-1 and database-2 would be started by that one cluster manager operation.

## Restrictions

- A resource group cannot contain an *equivalency* and an *equivalency* cannot contain a resource group (An *equivalency* is a set of resources that provide the same functionality as each other and can fail over for each other.)
- A resource can only be in one resource group
- A resource cannot be in a resource group and in an *equivalency*
- A resource group can contain other resource groups, but the maximum nesting level is 50
- The maximum number of resources that you can collect in a resource group is 100

*Quorum devices:*

A *quorum device* helps a cluster manager make cluster management decisions when the cluster manager's normal decision process does not produce a clear choice.

When a cluster manager has to choose between multiple potential actions, the cluster manager counts how many cluster domain nodes support each of the potential actions; and then cluster manager chooses the action that is supported by the majority of cluster domain nodes. If exactly the same number of cluster domain nodes supports more than one choice, then the cluster manager refers to a quorum device to make the choice.

**db2haicu** supports the quorum devices listed in the following table.

Table 4. Types of quorum device supported by **db2haicu**

Quorum device	Description
network	A network quorum device is an IP address to which every cluster domain node can connect at all times.

*Networks in a cluster domain:*

To configure elements of your cluster domain that are related to networks, you can use Db2 high availability instance configuration utility (**db2haicu**) to add a *physical network* to your cluster domain. A physical network is composed of: network interface cards, IP addresses, and subnetwork masks.

## Network interface cards

A *network interface card* (NIC) is hardware that connects a computer (also called a *cluster node*) to a network. A NIC is sometimes referred to as an *interface*, a *network adaptor*, or an *adaptor*. When you use **db2haicu** to add a physical network to your cluster domain, you specify at least one NIC including: the host name of the computer to which the NIC belongs; the name of the NIC on that host computer; and the IP address of the NIC.

## IP addresses

An *Internet Protocol address* (IP address) is a unique address on a network. In IP version 4, an IP address is 32 bits large, and is normally expressed in dot-decimal notation like this: 129.30.180.16. An IP address is composed of a network portion and a host computer portion.

## Subnetwork masks

A network can be partitioned into multiple logical subnetworks using *subnetwork masks*. A subnetwork mask is a mechanism for moving some bits of the host portion of an IP address to the network portion of the IP address. When you use **db2haicu** to add an IP address to your cluster domain, you will sometimes need to specify the subnetwork mask for the IP address. For example, when you use **db2haicu** to add a NIC, you must specify the subnetwork mask for the IP address of the NIC.

## Network equivalencies

An *equivalency* is a set of resources that provide the same functionality as each other and can fail over for each other. When you create a network using **db2haicu**, the NICs in that network can fail over for each other. Such a network is also referred to as a *network equivalency*.

## Network protocols

When you use **db2haicu** to add a network to your cluster domain, you must specify the type of network protocol being used. Currently, only the TCP/IP network protocol is supported.

## Usage note

A network configured using **db2haicu** is only required for a virtual IP (VIP) failover. Network adapters that are in different subnets (or equivalently, in different virtual local area networks) cannot be added to the same network because a common virtual local area network is required for a VIP failover.

*Failover policies in a cluster domain:*

A *failover policy* specifies how a cluster manager should respond when a cluster element such as a network interface card or a database server fails. In general, a cluster manager will transfer workload away from a failed element to an alternative element that had been previously identified to the cluster manager as an appropriate replacement for the element that failed. This transfer of workload from a failed element to a secondary element is called *failover*.

## Round robin failover policy

When you are using a *round robin failover policy*, if a failure occurs with one computer in the cluster domain (also called *cluster domain nodes* or *nodes*); the database manager restarts the work from the failed cluster domain node on any other node that is in the cluster domain. The *round robin failover policy* is available for both single and multiple database partition configurations.

## Mutual failover policy

To configure a *mutual failover policy*, you associate a pair of computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) as a system pair. If there is a failure on one of the nodes in this pair, then the database partitions on the failed node will failover to the other node in the pair. Mutual failover is only available when you have multiple database partitions.

## **N Plus M failover policy**

When you are using a *N Plus M failover policy*, then if there is a failure associated with one computer in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database partitions on the failed node will failover to any other node that is in the cluster domain. If roving HA failover is enabled, the last failed node becomes the standby node once that failed node is brought online again. The roving HA failover for N plus M failover policy is only supported for the case where M=1. N Plus M failover is only available when you have multiple database partitions.

## **Local restart failover policy**

When you use the *local restart failover policy* and a failure on one of the computers in the cluster domain (also called *cluster domain nodes* or *nodes*) occurs, the database manager restarts the database in place (or locally) on the same node that failed. The *local restart failover policy* is available for both single and multiple database partition configurations.

## **HADR failover policy**

When you configure a *HADR failover policy*, you are enabling the Db2 High Availability Disaster Recovery (HADR) feature to manage failover. If an HADR primary database fails, the database manager will move the workload from the failed database to an HADR standby database.

## **Custom failover policy**

When you configure a *custom failover policy*, you create a list of computers in the cluster domain (also called *cluster domain nodes* or *nodes*) onto which the database manager can failover. If a node in the cluster domain fails, the database manager moves the workload from the failed node to one of the nodes in the list that you specified. The *custom failover policy* is available for both single and multiple database partition configurations.

*Using roving high availability (HA) failover in partitioned database environments:*

When you are using a N Plus M failover policy with 'N' active nodes and one standby node, you can enable roving HA failover.

## **Before you begin**

Each node in the cluster must have the roving HA failover support enabled or disabled.

In partitioned database environments where roving HA failover is not enabled, the designated standby node is usually the only node with access to all the disks and volume groups, including the file systems on these volume groups. In those environments, ensure that the external storage LUN mappings and the SAN zones in the cluster can see all the disks in the database instance. In addition, verify that all the volume groups controlled by the cluster are imported on all the cluster nodes. After importing the volume groups, disable the auto-varyon attribute of volume groups and the auto-mount attribute of the file systems on all the active cluster nodes.

If you want to use roving HA failover, you must enable it again using these steps after applying a new fix pack.

### About this task

When you are using a *N Plus M* failover policy with 'N' active nodes and one standby node, a failover operation occurs when one of the active nodes fails. The standby node then begins hosting the resources of the failed node. When the failed node comes back online, you usually have to take the clustered environment offline again so the node which was originally chosen as the standby node becomes the standby node again. You can configure roving HA failover to have the last failed node in the cluster become the standby node without requiring additional fail back operations.

### Procedure

To enable roving HA failover:

1. Ensure that there is no failover operation in progress.
2. Make a backup copy of the `db2V10_start.ksh` script located in the `sqllib\samples\tsa` directory.
3. Edit the `db2V10_start.ksh` script. Find the following line:  
`ROVING_STANDBY_ENABLED=false`

and make the following changes:

`ROVING_STANDBY_ENABLED=true`

4. Save your changes.

### Results

The change will take effect at the next failover operation.

### What to do next

If you want to disable roving HA failover support, perform the following steps on each node:

1. Ensure that there is no failover operation in progress.
2. Edit the `db2V10_start.ksh` script. Find the following line:  
`ROVING_STANDBY_ENABLED=true`  
  
and make the following changes:  
`ROVING_STANDBY_ENABLED=false`
3. Save your changes. The change will take effect at the next failover operation.

*Mount points in a cluster domain:*

After mounting a file system, you can use Db2 high availability instance configuration utility (**db2haicu**) to add that mount point to your cluster domain.

### Mount points

On UNIX, Linux, and AIX operating systems, to *mount* a file system means to make that file system available to the operating system. During the mount operation, the operating system performs tasks such as reading index or navigation data structures, and associates a directory path with that mounted file system. That



associated directory path that you can use to access the mounted file system is called a *mount point*.

### Non-critical mount points or paths

There might be mount points or paths in your cluster that do not need to be failed over in the event of a failure. You can use **db2haicu** to add a list of those non-critical mount points or paths to your cluster domain. Your cluster manager will not include the mount points or paths in that non-critical list in failover operations.

For example, consider the case where you have a hard drive mounted at `/mnt/driveA` on a computer called `node1` in your cluster. If you decide that it is critical for `/mnt/driveA` to be available, your cluster manager will fail over to keep `/mnt/driveA` available if `node1` fails. However, if you decide that it is acceptable for `/mnt/driveA` to be unavailable if `node1` fails, then you can indicate to your cluster manager that `/mnt/driveA` is not critical for failover by adding `/mnt/driveA` to the list of non-critical paths. If `/mnt/driveA` is identified as non-critical for failover, then that drive might be unavailable if `node1` fails.

### Db2 high availability instance configuration utility (db2haicu):

Db2 high availability instance configuration utility (**db2haicu**) is a text-based utility that you can use to configure and administer your highly available databases in a clustered environment.

**db2haicu** collects information about your database instance, your cluster environment, and your cluster manager by querying your system. You supply more information through parameters to the **db2haicu** call, an input file, or at run time by providing information at **db2haicu** prompts.

#### Syntax

```
db2haicu [ -f XML-input-file-name ]
          [ -o XML-output-file-name ]
          [ -disable ]
          [ -delete [ dbpartitionnum db-partition-list |
                    hadrdb database-name ] ]
```

#### Parameters

The parameters that you pass to the **db2haicu** command are case-sensitive, and must be in lower case.

##### **-f XML-input-file-name**

You can use the **-f** parameter to specify your cluster domain details in an XML input file, *XML-input-file-name*. For more information, see: “Running Db2 High Availability Instance Configuration Utility (db2haicu) with an XML input file” on page 56.

##### **-o XML-output-file-name**

You can use the **-o** option to export the current TSAMP configuration to an XML output file. This is supported in all **db2haicu** deployed HA environments. In a HADR environment, the export can be performed on the primary and standby hosts. If the export is run on a single host (either primary or standby) and the same XML file is used to import the configuration later via the **-f** option, the **localHost** and **remoteHost** parameters of HADRDB element will need to be modified depending on the host where the import is being performed. In a DPF/Single partition

(non-DPF) environment, the user can export and/or import the XML configuration file from any host in the cluster. This option can be run with the instance offline or online.

#### **-disable**

A database manager instance is considered *configured for high availability* after you use **db2haicu** to create a cluster domain for that instance. When a database manager instance is configured for high availability, then whenever you perform certain database manager administrative operations that require related cluster configuration changes, the database manager communicates those cluster configuration changes to the cluster manager. When the database manager coordinates these cluster management tasks with the cluster manager for you, you do not have to perform a separate cluster manager operation for those administrative tasks. This integration between the database manager and the cluster manager is a function of the Db2 High Availability Feature.

You can use the **-disable** parameter to cause a database manager instance to cease to be configured for high availability. If the database manager instance is no longer configured for high availability, then the database manager does not coordinate with the cluster manager if you perform any database manager administrative operations that require related cluster configuration changes.

To reconfigure a database manager instance for high availability, you can run **db2haicu** again.

In an automated HADR environment, by running the **db2haicu -disable** command the database manager instance ceases to be configured for high availability on both the HADR primary and standby hosts. Subsequently by running **db2haicu**, high availability is re-enabled on both the HADR primary and standby hosts.

#### **-delete**

You can use the **-delete** parameter to delete resource groups for the current database manager instance.

If you do not use either the **dbpartitionnum** parameter or the **hadrdp** parameter, then **db2haicu** removes all the resource groups that are associated with the current database manager instance.

##### **dbpartitionnum db-partition-list**

You can use the **dbpartitionnum** parameter to delete resource groups that are associated with the database partitions listed in *db-partition-list*. *db-partition-list* is a comma-separated list of numbers that identify the database partitions.

##### **hadrdp database-name**

You can use the **hadrdp** parameter to delete resource groups that are associated with the high availability disaster recovery (HADR) database named *database-name*.

If there are no resource groups that are left in the cluster domain after **db2haicu** removes the resource groups, then **db2haicu** will also remove the cluster domain.

Running **db2haicu** with the **-delete** parameter causes the current database manager instance to cease to be configured for high availability. If the database manager instance is no longer configured for high availability,

then the database manager does not coordinate with the cluster manager if you perform any database manager administrative operations that require related cluster configuration changes.

To reconfigure a database manager instance for high availability, you can run **db2haicu** again.

#### *Db2 High Availability Instance Configuration Utility (db2haicu) startup mode:*

The first time that you run Db2 high availability instance configuration utility (**db2haicu**) for a given database manager instance, **db2haicu** operates in startup mode.

When you run **db2haicu**, **db2haicu** examines your database manager instance and your system configuration, and searches for an existing *cluster domain*. A cluster domain is a model that contains information about your cluster elements such as databases, mount points, and failover policies. You create a cluster domain using Db2 high availability instance configuration utility (**db2haicu**).

When you run **db2haicu** for a given database manager instance, and there is no cluster domain that is already created and configured for that instance, **db2haicu** will immediately begin the process of creating and configuring a new cluster domain. **db2haicu** creates a new cluster domain by prompting you for information such as a name for the new cluster domain and the hostname of the current machine.

If you create a cluster domain, but do not complete the task of configuring the cluster domain, then the next time you run **db2haicu**, **db2haicu** will resume the task of configuring the cluster domain.

After you create and configure a cluster domain for a database manager instance, **db2haicu** will run in maintenance mode.

#### *Db2 High Availability Instance Configuration Utility (db2haicu) maintenance mode:*

When you run Db2 high availability instance configuration utility (**db2haicu**) and there is already a cluster domain created for the current database manager instance, **db2haicu** operates in maintenance mode.

When **db2haicu** is running in maintenance mode, **db2haicu** presents you with a list of configuration and administration tasks that you can perform.

**db2haicu** maintenance tasks include adding cluster elements such as databases or cluster nodes to the cluster domain, and removing elements from the cluster domain. **db2haicu** maintenance tasks also include modifying the details of cluster domain elements such as the failover policy for the database manager instance.

When you run **db2haicu** in maintenance mode, **db2haicu** presents you with a list of operations you can perform on the cluster domain:

- Add or remove cluster nodes (machine identified by hostname)
- Add or remove a network interface (network interface card)
- Add or remove database partitions (partitioned database environment only)
- Add or remove a Db2 High Availability Disaster Recovery (HADR) database
- Add or remove a highly available database
- Add or remove a mount point

- Add or remove an IP address
- Add or remove a non-critical path
- Move database partitions and HADR databases for scheduled maintenance
- Change failover policy for the current instance
- Create a new quorum device for the cluster domain
- Destroy the cluster domain

*Running Db2 High Availability Instance Configuration Utility (db2haicu) in interactive mode:*

When you invoke Db2 high availability instance configuration utility (**db2haicu**) by running the **db2haicu** command without specifying an XML input file with the **-f** parameter, the utility runs in interactive mode. In interactive mode, **db2haicu** displays information and queries you for information in a text-based format.

#### **Before you begin**

- There is a set of tasks you must perform before using Db2 high availability instance configuration utility (**db2haicu**). For more information, see: “Db2 High Availability Instance Configuration Utility (db2haicu) prerequisites” on page 86.

#### **About this task**

When you run **db2haicu** in interactive mode, you see information and questions presented to you in text format on your screen. You can enter the information requested by **db2haicu** at the on-screen prompt.

#### **Procedure**

To run **db2haicu** in interactive mode, call the **db2haicu** command without the **-f input-file-name**.

#### **What to do next**

Db2 high availability instance configuration utility (**db2haicu**) does not have a separate diagnostic log. You can investigate and diagnose **db2haicu** errors using the database manager diagnostic log, **db2diag** log file, and the **db2pd** tool. For more information, see: “Troubleshooting Db2 High Availability Instance Configuration Utility (db2haicu)” on page 89

*Running Db2 High Availability Instance Configuration Utility (db2haicu) with an XML input file:*

You can use the **-f input-file-name** parameter with the **db2haicu** command to run Db2 high availability instance configuration utility (**db2haicu**) with an XML input file specifying your configuration details. Running **db2haicu** with an XML input file is useful when you must perform configuration tasks multiple times, such as when you have multiple database partitions to be configured for high availability.

#### **Before you begin**

- There is a set of tasks you must perform before using Db2 high availability instance configuration utility (**db2haicu**). For more information, see: “Db2 High Availability Instance Configuration Utility (db2haicu) prerequisites” on page 86.

## About this task

There is a set of sample XML input files located in the `samples` subdirectory of the `sqllib` directory that you can modify and use with **db2haicu** to configure your clustered environment. For more information, see: “Sample XML input files for Db2 High Availability Instance Configuration Utility (db2haicu)” on page 77

For a detailed scenario that uses **db2haicu** with a sample XML input file to set up an HADR pair, see “Automated Cluster Controlled HADR (High Availability Disaster Recovery) Configuration Setup using the IBM Db2 High Availability Instance Configuration Utility (db2haicu)”.

## Procedure

1. Create an XML input file. You will use the same XML file if you are configuring database partitions or, in an HADR setup, both the primary and the standby.
2. Call **db2haicu** with the `-f input-file-name`. In an HADR setup,
  - a. Log on to the standby instance and issue the command.
  - b. After **db2haicu** exits, log on to the primary instance and issue the command.

## What to do next

Db2 high availability instance configuration utility (**db2haicu**) does not have a separate diagnostic log. You can investigate and diagnose **db2haicu** errors using the database manager diagnostic log, **db2diag** log file, and the **db2pd** tool. For more information, see: “Troubleshooting Db2 High Availability Instance Configuration Utility (db2haicu)” on page 89

*Db2 High Availability Instance Configuration Utility (db2haicu) input file XML schema definition:*

The Db2 high availability instance configuration utility (**db2haicu**) input file XML schema definition (XSD) defines the cluster domain objects that you can specify in a **db2haicu** XML input file. This **db2haicu** XSD is located in the file called `db2ha.xsd` in the `sqllib/samples/ha/xml` directory.

## DB2ClusterType

The root element of the **db2haicu** XML schema definition (XSD) is `DB2Cluster`, which is of type `DB2ClusterType`. A **db2haicu** XML input file must begin with a `DB2Cluster` element.

“XML schema definition”

“Subelements” on page 58

“Attributes” on page 59

“Usage notes” on page 59

## XML schema definition

```
<xs:complexType name='DB2ClusterType'>
  <xs:sequence>
    <xs:element name='DB2ClusterTemplate'
      type='DB2ClusterTemplateType'
      minOccurs='0'
      maxOccurs='unbounded' />
    <xs:element name='ClusterDomain'
      type='ClusterDomainType'
      maxOccurs='unbounded' />
  </xs:sequence>
</xs:complexType>
```

```

<xs:element name='FailoverPolicy'
  type='FailoverPolicyType'
  minOccurs='0' />
<xs:element name='DB2PartitionSet'
  type='DB2PartitionSetType'
  minOccurs='0'
  maxOccurs='unbounded' />
<xs:element name='HADRDBSet'
  type='HADRDBType'
  minOccurs='0'
  maxOccurs='unbounded' />
<xs:element name='HADBSet'
  type='HADBType'
  minOccurs='0'
  maxOccurs='unbounded' />
</xs:sequence>
<xs:attribute name='clusterManagerName' type='xs:string' use='optional' />
</xs:complexType>

```

## Subelements

### DB2ClusterTemplate

**Type:** DB2ClusterTemplateType

**Usage notes:**

Do not include a DB2ClusterTemplateType element in your **db2haicu** XML input file. The DB2ClusterTemplateType element is currently reserved for future use.

### ClusterDomain

**Type:** ClusterDomainType

A ClusterDomainType element contains specifications about: the machines or computers in the cluster domain (also called *cluster domain nodes*); the *network equivalencies* (groups of networks that can fail over for one another); and the *quorum device* (tie-breaking mechanism).

**Occurrence rules:**

You must include one or more ClusterDomain element in your DB2ClusterType element.

### FailoverPolicy

**Type:** FailoverPolicyType

A FailoverPolicyType element specifies the *failover policy* that the cluster manager should use with the cluster domain.

**Occurrence rules:**

You can include zero or one FailoverPolicy element in your DB2ClusterType element.

### DB2PartitionSet

**Type:** DB2PartitionSetType

A DB2PartitionSetType element contains information about database partitions. The DB2PartitionSetType element is only applicable in a partitioned database environment.

**Occurrence rules:**

You can include zero or more DB2PartitionSet elements in your DB2ClusterType element, according to the **db2haicu** db2haicu XML schema definition.

**HADRDBSet**

**Type:** HADRDBType

A HADRDBType element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

**Occurrence rules:**

You can include zero or more HADRDBSet elements in your DB2ClusterType element, according to the **db2haicu** db2haicu XML schema definition.

**Usage notes:**

- You must not include HADRDBSet in a partitioned database environment.
- If you include HADRDBSet, then you must specify a failover policy of HADRFailover in the FailoverPolicy element.

**HADBSet**

**Type:** HADBType

A HADBType element contains a list of databases to include in the cluster domain, and to make highly available.

**Occurrence rules:**

You can include zero or more HADBSet elements in your DB2ClusterType element, according to the **db2haicu** db2haicu XML schema definition.

**Attributes****clusterManagerName (optional)**

The clusterManagerName attribute specifies the cluster manager.

Valid values for this attribute are specified in the following table:

*Table 5. Valid values for the clusterManager attribute*

<b>clusterManagerName value</b>	<b>Cluster manager product</b>
TSA	IBM Tivoli System Automation for Multiplatforms (SA MP)

**Usage notes**

In a single partition database environment, you will usually only create a single cluster domain for each database manager instance.

One possible configuration for a multi-partition database environment is:

- Set the FailoverPolicy element to Mutual
- In the DB2Partition subelement of DB2PartitionSet, use the MutualPair element to specify two cluster domain nodes that are in a single cluster domain

*ClusterDomainType* XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:

A *ClusterDomainType* element contains specifications about: the machines or computers in the cluster domain (also called *cluster domain nodes*); the *network equivalencies* (groups of networks that can fail over for one another); and the *quorum device* (tie-breaking mechanism).

“Superelements”  
“XML schema definition”  
“Subelements”  
“Attributes” on page 61

## Superelements

The following types of elements contain *ClusterDomainType* subelements:

- *DB2ClusterType*

## XML schema definition

```
<xs:complexType name='ClusterDomainType'>
  <xs:sequence>
    <xs:element name='Quorum'
      type='QuorumType'
      minOccurs='0' />
    <xs:element name='PhysicalNetwork'
      type='PhysicalNetworkType'
      minOccurs='0'
      maxOccurs='unbounded' />
    <xs:element name='ClusterNode'
      type='ClusterNodeType'
      maxOccurs='unbounded' />
  </xs:sequence>
  <xs:attribute name='domainName' type='xs:string' use='required' />
</xs:complexType>
```

## Subelements

### Quorum

**Type:** *QuorumType*

A *QuorumType* element specifies the *quorum device* for the cluster domain.

**Occurrence rules:**

You can include zero or one *Quorum* element in your *ClusterDomainType* element.

### PhysicalNetwork

**Type:** *PhysicalNetworkType*

A *PhysicalNetworkType* element contains network interface cards that can fail over for each other. This kind of network is also called a *network equivalency*.

**Occurrence rules:**

You can include zero or more *PhysicalNetwork* elements in your *ClusterDomainType* element.

### ClusterNode

**Type:** *ClusterNodeType*



A `ClusterNodeType` element contains information about a particular computer or machine (also called a *cluster domain node*) in the cluster.

**Occurrence rules:**

You must specify at least one `ClusterNode` element in your `ClusterDomainType` element.

**Usage notes**

IBM Tivoli System Automation for Multiplatforms (SA MP) supports a maximum of 32 cluster domain nodes. If your cluster manager is SA MP, then you can include a maximum of 32 `ClusterNode` elements in your `ClusterDomainType` element.

**Attributes**

**domainName (required)**

You must specify a unique name for your `ClusterDomainType` element.

If you are using Reliable Scalable Cluster Technology (RSCT) to manage your cluster, the following restrictions apply to `domainName`:

- `domainName` can only contain the characters A to Z, a to z, digits 0 to 9, period (.), and underscore (\_)
- `domainName` cannot be "IW"

The following example is of a `ClusterDomainType` element:

```
<ClusterDomain domainName="hadr_linux_domain">
  <Quorum quorumDeviceProtocol="network" quorumDeviceName="9.26.4.5"/>
  <PhysicalNetwork physicalNetworkName="db2_public_network_0"
    physicalNetworkProtocol="ip">
    <Interface interfaceName="eth0" clusterNodeName="linux01">
      <IPAddress baseAddress="9.26.124.30" subnetMask="255.255.255.0"
        networkName="db2_public_network_0"/>
    </Interface>
    <Interface interfaceName="eth0" clusterNodeName="linux02">
      <IPAddress baseAddress="9.26.124.31" subnetMask="255.255.255.0"
        networkName="db2_public_network_0"/>
    </Interface>
  </PhysicalNetwork>
  <ClusterNode clusterNodeName="linux01"/>
  <ClusterNode clusterNodeName="linux02"/>
</ClusterDomain>
```

*QuorumType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A `QuorumType` element specifies the *quorum device* for the cluster domain.

“Superelements”

“XML schema definition” on page 62

“Subelements” on page 62

“Attributes” on page 62

**Superelements**

The following types of elements contain `QuorumType` subelements:

- `ClusterDomainType`

## XML schema definition

```
<xs:complexType name='QuorumType'>
  <xs:attribute name='quorumDeviceProtocol'
    type='QuorumDeviceProtocolType'
    use='required' />
  <xs:attribute name='quorumDeviceName'
    type='xs:string'
    use='required' />
</xs:complexType>
```

## Subelements

None.

## Attributes

### quorumDeviceProtocol (required)

quorumDeviceProtocol specifies the type of quorum to use.

A *quorum device* helps a cluster manager make cluster management decisions when the cluster manager's normal decision process does not produce a clear choice.

The type of the quorumDeviceProtocol attribute is QuorumDeviceProtocolType.

Here is the XML schema definition for the QuorumDeviceProtocolType:

```
<xs:simpleType name='QuorumDeviceProtocolType'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='disk' />
    <xs:enumeration value='scsi' />
    <xs:enumeration value='network' />
    <xs:enumeration value='eckd' />
    <xs:enumeration value='mns' />
  </xs:restriction>
</xs:simpleType>
```

Currently supported values for this attribute are specified in the following table:

Table 6. Valid values for the quorumDeviceProtocol attribute

quorumDeviceProtocol value	Meaning
network	A network quorum device is an IP address to which every cluster domain node can connect at all times.

### quorumDeviceName (required)

The value of the quorumDeviceName depends on the type of quorum device specified in quorumDeviceProtocol.

Valid values for this attribute are specified in the following table:

Table 7. Valid values for the *quorumDeviceName* attribute

Value of <i>quorumDeviceProtocol</i>	Valid value for <i>quorumDeviceName</i>
network	<p>A string containing a properly formatted IP address. For example:</p> <p>12.126.4.5</p> <p>For the IP address that you specify to be valid as a network quorum device, every cluster domain node must be able to access this IP addressed (using the ping utility, for example.)</p>

*PhysicalNetworkType* XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:

A *PhysicalNetworkType* element contains network interface cards that can fail over for each other. This kind of network is also called a *network equivalency*.

“Superelements”

“XML schema definition”

“Subelements”

“Attributes” on page 64

## Superelements

The following types of elements contain *PhysicalNetworkType* subelements:

- *ClusterDomainType*

## XML schema definition

```
<xs:complexType name='PhysicalNetworkType'>
  <xs:sequence>
    <xs:element name='Interface'
      type='InterfaceType'
      minOccurs='1'
      maxOccurs='unbounded' />
    <xs:element name='LogicalSubnet'
      type='IPAddressType'
      minOccurs='0'
      maxOccurs='unbounded' />
  </xs:sequence>
  <xs:attribute name='physicalNetworkName'
    type='xs:string'
    use='required' />
  <xs:attribute name='physicalNetworkProtocol'
    type='PhysicalNetworkProtocolType'
    use='required' />
</xs:complexType>
```

## Subelements

### Interface

**Type:** *InterfaceType*

The *InterfaceType* element consists of an IP address, the name of a computer or machine in the network (also called a *cluster domain node*), and the name of a *network interface card* (NIC) on that cluster domain node.

**Occurrence rules:**

You must specify one or more Interface elements in your PhysicalNetworkType element.

**LogicalSubnet**

**Type:** IPAddressType

A IPAddressType element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

**Occurrence rules:**

You can include zero or more LogicalSubnet elements in your PhysicalNetworkType element.

**Attributes****physicalNetworkName (required)**

You must specify a unique physicalNetworkName for each PhysicalNetworkType element.

**physicalNetworkProtocol (required)**

The type of the physicalNetworkProtocol attribute is PhysicalNetworkProtocolType.

Here is the XML schema definition for the PhysicalNetworkProtocolType element:

```
<xs:simpleType name='PhysicalNetworkProtocolType'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='ip' />
    <xs:enumeration value='rs232' />
    <xs:enumeration value='scsi' />
    <xs:enumeration value='ssa' />
    <xs:enumeration value='disk' />
  </xs:restriction>
</xs:simpleType>
```

Currently supported values for this attribute are specified in the following table:

Table 8. Valid values for the physicalNetworkProtocol attribute

physicalNetworkProtocol value	Meaning
ip	TCP/IP protocol

*InterfaceType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

The InterfaceType element consists of an IP address, the name of a computer or machine in the network (also called a *cluster domain node*), and the name of a *network interface card* (NIC) on that cluster domain node.

“Superelements” on page 65

“XML schema definition” on page 65

“Subelements” on page 65

“Attributes” on page 65

## Superelements

The following types of elements have InterfaceType subelements:

- PhysicalNetworkType

### XML schema definition

```
<xs:complexType name='InterfaceType'>
  <xs:sequence>
    <xs:element name='IPAddress' type='IPAddressType' />
  </xs:sequence>
  <xs:attribute name='interfaceName' type='xs:string' use='required' />
  <xs:attribute name='clusterNodeName' type='xs:string' use='required' />
</xs:complexType>
```

## Subelements

### IPAddress

**Type:** IPAddressType

A IPAddressType element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

**Occurrence rules:**

You must specify exactly one IPAddress in your InterfaceType element.

## Attributes

### interfaceName (required)

You must specify the name of a NIC in the interfaceName attribute. The NIC that you specify in the interfaceName must exist on the cluster domain node that you specify in the clusterNodeName attribute.

### clusterNodeName (required)

You must specify the name of the cluster domain node that is located at the IP address that you specify in the IPAddress element.

*IPAddressType XML schema element for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A IPAddressType element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

“Superelements”

“XML schema definition” on page 66

“Subelements” on page 66

“Attributes” on page 66

## Superelements

The following types of elements have IPAddressType subelements:

- PhysicalNetworkType
- InterfaceType
- DB2PartitionType

### XML schema definition

```
<xs:complexType name='IPAddressType'>
  <xs:attribute name='baseAddress' type='xs:string' use='required' />
  <xs:attribute name='subnetMask' type='xs:string' use='required' />
  <xs:attribute name='networkName' type='xs:string' use='required' />
</xs:complexType>
```

### Subelements

None.

### Attributes

#### baseAddress (required)

You must specify the base IP address using a string with a valid IP address format: four sets of numbers ranging from 0 to 255, separated by a period. For example:

162.148.31.101

#### subnetMask (required)

You must specify the base IP address using a string with a valid IP address format.

#### networkName (required)

You must specify the same value for networkName here as you specified for the physicalNetworkName attribute of the PhysicalNetworkType element that contains this IPAddress element.

*ClusterNodeType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A ClusterNodeType element contains information about a particular computer or machine (also called a *cluster domain node*) in the cluster.

“Superelements”

“XML schema definition”

“Subelements”

“Attributes”

### Superelements

The following types of elements have ClusterNodeType elements:

- ClusterDomainType

### XML schema definition

```
<xs:complexType name='ClusterNodeType'>
  <xs:attribute name='clusterNodeName' type='xs:string' use='required' />
</xs:complexType>
```

### Subelements

None.

### Attributes

#### clusterNodeName (required)

You must specify the name of the cluster domain node.

*FailoverPolicyType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A `FailoverPolicyType` element specifies the *failover policy* that the cluster manager should use with the cluster domain.

“Superelements”

“XML schema definition”

“Subelements”

“Possible values”

## Superelements

The following types of elements contain `InterfaceType` subelements:

- `DB2ClusterType`

## XML schema definition

```
<xs:complexType name='FailoverPolicyType'>
  <xs:choice>
    <xs:element name='RoundRobin'
      type='xs:string'
      minOccurs='0' />
    <xs:element name='Mutual'
      type='xs:string'
      minOccurs='0'
      maxOccurs='unbounded' />
    <xs:element name='NPlusM'
      type='xs:string'
      minOccurs='0'
      maxOccurs='unbounded' />
    <xs:element name='LocalRestart'
      type='xs:string'
      fixed='1' />
    <xs:element name='HADRFailover'
      type='xs:string'
      fixed='1' />
    <xs:element name='Custom'
      type='xs:string'
      minOccurs='0' />
  </xs:choice>
</xs:complexType>
```

## Subelements

None.

## Possible values

Select one of the following choices to specify to the cluster manager what type of failover policy to use if there is a failure anywhere in the cluster domain.

A *failover policy* specifies how a cluster manager should respond when a cluster element such as a network interface card or a database server fails. In general, a cluster manager will transfer workload away from a failed element to an alternative element that had been previously identified to the cluster manager as an appropriate replacement for the element that failed. This transfer of workload from a failed element to a secondary element is called *failover*.

## RoundRobin

When you are using a *round robin failover policy*, if a failure occurs with one

computer in the cluster domain (also called *cluster domain nodes* or *nodes*); the database manager restarts the work from the failed cluster domain node on any other node that is in the cluster domain. The *round robin failover policy* is available for both single and multiple database partition configurations.

#### **Mutual**

To configure a *mutual failover policy*, you associate a pair of computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) as a system pair. If there is a failure on one of the nodes in this pair, then the database partitions on the failed node will failover to the other node in the pair. Mutual failover is only available when you have multiple database partitions.

#### **NPlusM**

When you are using a *N Plus M failover policy*, then if there is a failure associated with one computer in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database partitions on the failed node will failover to any other node that is in the cluster domain. If roving HA failover is enabled, the last failed node become the standby node once that failed node is brought online again. The roving HA failover for N plus M failover policy is only supported for the case where M=1. N Plus M failover is only available when you have multiple database partitions.

#### **LocalRestart**

When you use the *local restart failover policy* and a failure on one of the computers in the cluster domain (also called *cluster domain nodes* or *nodes*) occurs, the database manager restarts the database in place (or locally) on the same node that failed. The *local restart failover policy* is available for both single and multiple database partition configurations.

#### **HADRFailover**

When you configure a *HADR failover policy*, you are enabling the Db2 High Availability Disaster Recovery (HADR) feature to manage failover. If an HADR primary database fails, the database manager will move the workload from the failed database to an HADR standby database.

#### **Custom**

When you configure a *custom failover policy*, you create a list of computers in the cluster domain (also called *cluster domain nodes* or *nodes*) onto which the database manager can failover. If a node in the cluster domain fails, the database manager moves the workload from the failed node to one of the nodes in the list that you specified. The *custom failover policy* is available for both single and multiple database partition configurations.

*DB2PartitionSetType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A DB2PartitionSetType element contains information about database partitions. The DB2PartitionSetType element is only applicable in a partitioned database environment.

“Superelements” on page 69

“XML schema definition” on page 69

“Subelements” on page 69

“Attributes” on page 69



## Superelements

InterfaceType is a subelement of:

- PhysicalNetworkType

## XML schema definition

```
<xs:complexType name='DB2PartitionSetType'>
  <xs:sequence>
    <xs:element name='DB2Partition'
      type='DB2PartitionType'
      maxOccurs='unbounded' />
  </xs:sequence>
</xs:complexType>
```

## Subelements

### DB2Partition

**Type:** DB2PartitionType

A DB2PartitionType element specifies a database partition including the Db2 database manager instance to which the database partition belongs and the database partition number.

**Occurrence rules:**

You must specify one or more DB2Partition elements in your DB2PartitionSetType element.

## Attributes

None.

*DB2PartitionType XML schema element for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A DB2PartitionType element specifies a database partition including the Db2 database manager instance to which the database partition belongs and the database partition number.

“Superelements”

“XML schema definition”

“Subelements” on page 70

“Attributes” on page 71

## Superelements

InterfaceType is a subelement of:

- DB2PartitionSetType

## XML schema definition

```
<xs:complexType name='DB2PartitionType'>
  <xs:sequence>
    <xs:element name='VirtualIPAddress'
      type='IPAddressType'
      minOccurs='0'
      maxOccurs='unbounded' />
    <xs:element name='Mount'
      type='MountType'
      minOccurs='0'
      maxOccurs='unbounded' />
  </xs:sequence>
</xs:complexType>
```

```

<xs:element name='HADRDB'
            type='HADRDBType'
            minOccurs='0'
            maxOccurs='unbounded' />
<xs:element name='MutualPair'
            type='MutualPolicyType'
            minOccurs='0'
            maxOccurs='1' />
<xs:element name='NPlusMNode'
            type='NPlusMPolicyType'
            minOccurs='0'
            maxOccurs='unbounded' />
<xs:element name='CustomNode'
            type='CustomPolicyType'
            minOccurs='0'
            maxOccurs='unbounded' />
</xs:sequence>
<xs:attribute name='instanceName'      type='xs:string'   use='required' />
<xs:attribute name='dbpartitionnum'    type='xs:integer' use='required' />
</xs:complexType>

```

## Subelements

### VirtualIPAddress

Type: IPAddressType

A IPAddressType element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

You can omit including VirtualIPAddress; or you can include an unbounded number of VirtualIPAddress elements in your DB2PartitionType element.

### Mount

Type: MountType

A MountType element contains information about a *mount point* such as the file path that identifies the location of the mounted files.

You can omit including Mount; or you can include an unbounded number of Mount elements in your DB2PartitionType element.

There is no XML element to specify non-critical mount points in DPF/single-partition setup. If the user intends to set up non-critical mount points, they would need to specify all critical mount points using the MountType element. Any element not included in this list will be considered non-critical. If this element is not specified, then no mount points will be considered as non-critical.

### HADRDB

Type: HADRDBType

A HADRDBType element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

You can omit including HADRDB; or you can include an unbounded number of HADRDB elements in your DB2PartitionType element.

### MutualPair

Type: MutualPolicyType

A MutualPolicyType element contains information about a pair of cluster domain nodes that can failover for each other.

You can omit including `MutualPair`; or you can include exactly one `MutualPair` elements in your `DB2PartitionType` element.

#### **NPlusMNode**

Type: `NPlusMPolicyType`

You can omit including `NPlusMNode`; or you can include an unbounded number of `NPlusMNode` elements in your `DB2PartitionType` element.

#### **CustomNode**

Type: `CustomPolicyType`

You can omit including `CustomNode`; or you can include an unbounded number of `CustomNode` elements in your `DB2PartitionType` element.

#### **Attributes**

##### **instanceName (required)**

In the `instanceName` attribute you must specify the Db2 database manager instance with which this `DB2PartitionType` element is associated.

##### **dbpartitionnum (required)**

In the `dbpartitionnum` attribute you must specify the database partition number that uniquely identifies the database partition (the `dbpartitionnum` number specified in the `db2nodes.cfg` file, for example.)

*MountType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A `MountType` element contains information about a *mount point* such as the file path that identifies the location of the mounted files.

“Superelements”

“XML schema definition”

“Subelements”

“Attributes”

#### **Superelements**

The following types of elements contain `MountType` subelements:

- `DB2PartitionType`

#### **XML schema definition**

```
<xs:complexType name='MountType'>
  <xs:attribute name='filesystemPath' type='xs:string' use='required' />
</xs:complexType>
```

#### **Subelements**

None.

#### **Attributes**

##### **filesystemPath (required)**

Specify the path that was associated with the mount point when the file system was mounted.

*MutualPolicyType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A `MutualPolicyType` element contains information about a pair of cluster domain nodes that can failover for each other.

“Superelement”  
“XML schema definition”  
“Subelements”  
“Attributes”

### Superelement

The following types of elements contain `MutualPolicyType` subelements:

- `DB2PartitionType`

### XML schema definition

```
<xs:complexType name='MutualPolicyType'>  
  <xs:attribute name='systemPairNode1' type='xs:string' use='required' />  
  <xs:attribute name='systemPairNode2' type='xs:string' use='required' />  
</xs:complexType>
```

### Subelements

None.

### Attributes

#### **systemPairNode1 (required)**

In `systemPairNode1` you must specify the name of a cluster domain node that can fail over for the cluster domain node that you specify in `systemPairNode2`.

#### **systemPairNode2 (required)**

In `systemPairNode2` you must specify the name of a cluster domain node that can fail over for the cluster domain node that you specify in `systemPairNode1`.

*NPlusMPolicyType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

An `NPlusMPolicy` states that if a computer in a cluster domain experiences a failure, then the database partitions on the failed node fails over to any other available node in the same cluster domain. An XML schema defines the configurations associated with this HADR policy.

“Superelements”  
“XML schema definition” on page 73  
“Subelements” on page 73  
“Attributes” on page 73

### Superelements

The following types of elements contain `NPlusMPolicyType` subelements:

- `DB2PartitionType`

### XML schema definition

```
<xs:complexType name='NPlusMPolicyType'>
  <xs:attribute name='standbyNodeName' type='xs:string' use='required' />
</xs:complexType>
```

### Subelements

None.

### Attributes

#### standbyNodeName (required)

In the standbyNodeName element, you must specify the name of a cluster domain node to which the partition that contains this NPlusMPolicyType element can fail over.

*CustomPolicyType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A CustomPolicyType XML schema defines configuration settings for a Custom HADR policy. You can define the nodes that a failover defaults to in this schema.

“Superelements”

“XML schema definition”

“Subelements”

“Attributes”

### Superelements

The following types of elements contain CustomPolicyType subelements:

- DB2PartitionType

### XML schema definition

```
<xs:complexType name='CustomNode'>
  <xs:attribute name='customNodeName' type='xs:string' use='required' />
</xs:complexType>
```

### Subelements

None.

### Attributes

#### customNodeName (required)

In the customNodeName element, you must specify the name of a cluster domain node to which the partition that contains this CustomPolicyType element can fail over.

*HADRDBType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A HADRDBType element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

“Superelements” on page 74

“XML schema definition” on page 74

“Subelements” on page 74

“Attributes”  
“Usage notes”  
“Restrictions”

## Superelements

The following types of elements contain HADRDBType subelements:

- DB2ClusterType
- DB2PartitionType

## XML schema definition

```
<xs:complexType name='HADRDBType'>
  <xs:sequence>
    <xs:element name='HADRDB' type='HADRDBDefn' minOccurs='1' maxOccurs='1' />
    <xs:element name='VirtualIPAddress' type='IPAddressType' minOccurs='0' maxOccurs='1' />
  </xs:sequence>
</xs:complexType>
```

## Subelements

### VirtualIPAddress

**Type:** IPAddressType

A IPAddressType element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

**Occurrence rules:**

You can including zero or more VirtualIPAddress elements in your HADRDBType element.

### HADRDB

**Type:** HADRDBDefn

A HADRDBDefn element contains information about a High Availability Disaster Recovery (HADR) primary and standby database pair.

**Occurrence rules:**

You can include one or more VirtualIPAddress elements in your HADRDBType element.

## Attributes

None.

## Usage notes

If you include a HADRDBType element in the specification for a given cluster domain, then you must also include a FailoverPolicy element specifying HADRFailover in the same cluster domain specification.

## Restrictions

You cannot use the HADRDBType element in a partitioned database environment.

The following example is of an HADRDBType element:

```

<HADRDBSet>
  <HADRDB databaseName="HADRDB" localInstance="db2inst1"
    remoteInstance="db2inst1" localHost="linux01" remoteHost="linux02" />
  <VirtualIPAddress baseAddress="9.26.124.22" subnetMask="255.255.245.0"
    networkName="db2_public_network_0"/>
</HADRDBSet>

```

*HADRDBDefn XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A HADRDBDefn element contains information about a High Availability Disaster Recovery (HADR) primary and standby database pair.

“Superelements”

“XML schema definition”

“Subelements”

“Attributes”

## Superelements

The following types of elements contain HADRDBDefn subelements:

- HADRDBType

## XML schema definition

```

<xs:complexType name='HADRDBDefn'>
  <xs:attribute name='databaseName' type='xs:string' use='required' />
  <xs:attribute name='localInstance' type='xs:string' use='required' />
  <xs:attribute name='remoteInstance' type='xs:string' use='required' />
  <xs:attribute name='localHost' type='xs:string' use='required' />
  <xs:attribute name='remoteHost' type='xs:string' use='required' />
</xs:complexType>

```

## Subelements

None.

## Attributes

### databaseName (required)

Enter the name of the HADR database.

### localInstance (required)

The localInstance is the database manager instance of the HADR primary database.

### remoteInstance (required)

The remoteInstance is the database manager instance of the HADR standby database.

### localHost (required)

The localHost is the hostname of the cluster domain node where the HADR primary database is located.

### remoteHost (required)

The remoteHost is the hostname of the cluster domain node where the HADR standby database is located.

*HADBType XML schema definition for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A HADBType element contains a list of databases to include in the cluster domain, and to make highly available.

“Superelements”  
“XML schema definition”  
“Subelements”  
“Attributes”

## Superelements

The following types of elements contain HADBType subelements:

- DB2ClusterType

## XML schema definition

```
<xs:complexType name='HADBType'>
  <xs:sequence>
    <xs:element name='HADB' type='HADBDefn' maxOccurs='unbounded' />
  </xs:sequence>
  <xs:attribute name='instanceName' type='xs:string' use='required' />
</xs:complexType>
```

## Subelements

### HADB

**Type:** HADBDefn

A HADBDefn element describes a database to be included in the cluster domain and made highly available.

**Occurrence rules:**

You must include one or more HADB elements in your HADBType element.

## Attributes

### instanceName (required)

In the instanceName attribute, you must specify the Db2 database manager instance to which the databases specified in the HADB elements belong.

*HADBDefn XML schema element for Db2 High Availability Instance Configuration Utility (db2haicu) input files:*

A HADBDefn element describes a database to be included in the cluster domain and made highly available.

“Superelements”  
“XML schema definition” on page 77  
“Subelements” on page 77  
“Attributes” on page 77

## Superelements

HADBDefn is a subelement of:

- HADRDBType



## XML schema definition

```
<xs:complexType name='HADBDfn'>
  <xs:attribute name='databaseName' type='xs:string' use='required' />
</xs:complexType>
```

## Subelements

None.

## Attributes

### databaseName (required)

You must specify exactly one database name in the databaseName attribute.

*Sample XML input files for Db2 High Availability Instance Configuration Utility (db2haicu):*

There is a set of sample XML input files located in the `samples` subdirectory of the `sqllib` directory that you can modify and use with **db2haicu** to configure your clustered environment.

*db2ha\_sample\_sharedstorage\_mutual.xml:*

The sample file `db2ha_sample_sharedstorage_mutual.xml` is an example of an XML input file that you pass to Db2 high availability instance configuration utility (**db2haicu**) to specify a new *cluster domain*.

`db2ha_sample_sharedstorage_mutual.xml` is located in the `sqllib/samples/ha/xml` directory.

## Features

The `db2ha_sample_sharedstorage_mutual.xml` sample demonstrates how to use **db2haicu** with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): two
- failover policy: mutual
- database partitions: one
- virtual (service) IP addresses: one
- shared mount points for failover: one

## XML source

```
<!-- ===== -->
<!-- = Use the Db2 High Availability Instance Configuration Utility = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP) = -->
<!-- = Base Component as the cluster manager. = -->
<!-- ===== -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="db2ha.xsd"
  clusterManagerName="TSA"
  version="1.0">

  <!-- ===== -->
  <!-- = Create a cluster domain named db2HADomain. = -->
  <!-- ===== -->
  <ClusterDomain domainName="db2HADomain">
```

```

<!-- ===== -->
<!-- = Specify a network quorum device (IP address: 19.126.4.5). = -->
<!-- = The IP must be pingable at all times by each of the cluster = -->
<!-- = domain nodes. = -->
<!-- ===== -->
<Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

<!-- ===== -->
<!-- = Create a network named db2_public_network_0 with an IP = -->
<!-- = network protocol. = -->
<!-- = This network contains two computers: hasys01 and hasys02. = -->
<!-- = Each computer has one network interface card (NIC) called = -->
<!-- = eth0. = -->
<!-- = The IP address of the NIC on hasys01 is 19.126.52.139 = -->
<!-- = The IP address of the NIC on hasys02 is 19.126.52.140 = -->
<!-- ===== -->
<PhysicalNetwork physicalNetworkName="db2_public_network_0"
    physicalNetworkProtocol="ip">

    <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.52.139"
            subnetMask="255.255.255.0"
            networkName="db2_public_network_0"/>
    </Interface>

    <Interface interfaceName="eth0" clusterNodeName="hasys02">
        <IPAddress baseAddress="19.126.52.140"
            subnetMask="255.255.255.0"
            networkName="db2_public_network_0"/>
    </Interface>

</PhysicalNetwork>

<!-- ===== -->
<!-- = List the computers (cluster nodes) in the cluster domain. = -->
<!-- ===== -->
<ClusterNode clusterNodeName="hasys01"/>
<ClusterNode clusterNodeName="hasys02"/>

</ClusterDomain>

<!-- ===== -->
<!-- = The failover policy specifies the order in which the cluster = -->
<!-- = domain nodes should fail over. = -->
<!-- ===== -->
<FailoverPolicy>
    <Mutual />
</FailoverPolicy>

<!-- ===== -->
<!-- = Specify all the details of the database partition = -->
<!-- ===== -->
<DB2PartitionSet>

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
        <VirtualIPAddress baseAddress="19.126.52.222"
            subnetMask="255.255.255.0"
            networkName="db2_public_network_0"/>
        <Mount filesystemPath="/home/db2inst1"/>
        <MutualPair systemPairNode1="hasys01" systemPairNode2="hasys02" />
    </DB2Partition>

</DB2PartitionSet>

</DB2Cluster>

```

*db2ha\_sample\_DPF\_mutual.xml:*

The sample file `db2ha_sample_DPF_mutual.xml` is an example of an XML input file that you pass to Db2 high availability instance configuration utility (**db2haicu**) to specify a new *cluster domain*. `db2ha_sample_DPF_mutual.xml` is located in the `sqllib/samples/ha/xml` directory.

## Features

The `db2ha_sample_DPF_mutual.xml` sample demonstrates how to use **db2haicu** with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): four
- failover policy: mutual
- database partitions: two
- virtual (service) IP addresses: one
- shared mount points for failover: two
- databases configured for high availability: two

## XML source

```
<!-- ===== -->
<!-- = Use the Db2 High Availability Instance Configuration Utility = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP) = -->
<!-- = Base Component as the cluster manager. = -->
<!-- ===== -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="db2ha.xsd"
             clusterManagerName="TSA"
             version="1.0">

  <!-- ===== -->
  <!-- = Create a cluster domain named db2HADomain. = -->
  <!-- ===== -->
  <ClusterDomain domainName="db2HADomain">

    <!-- ===== -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5). = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes. = -->
    <!-- ===== -->
    <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

    <!-- ===== -->
    <!-- = Create a network named db2_public_network_0 with an IP = -->
    <!-- = network protocol. = -->
    <!-- = This network contains four computers: hasys01, hasys02, = -->
    <!-- = hasys03, and hasys04. = -->
    <!-- = Each computer has a network interface card called eth0. = -->
    <!-- = The IP address of eth0 on hasys01 is 19.126.124.30 = -->
    <!-- = The IP address of eth0 on hasys02 is 19.126.124.31 = -->
    <!-- = The IP address of eth0 on hasys03 is 19.126.124.32 = -->
    <!-- = The IP address of eth0 on hasys04 is 19.126.124.33 = -->
    <!-- ===== -->
    <PhysicalNetwork physicalNetworkName="db2_public_network_0"
                    physicalNetworkProtocol="ip">

      <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.124.30"
                    subnetMask="255.255.255.0"
                    networkName="db2_public_network_0"/>
      </Interface>
    </PhysicalNetwork>
  </ClusterDomain>
</DB2Cluster>
```

```

</Interface>

<Interface interfaceName="eth0" clusterNodeName="hasys02">
  <IPAddress baseAddress="19.126.124.31"
    subnetMask="255.255.255.0"
    networkName="db2_public_network_0"/>
</Interface>

<Interface interfaceName="eth0" clusterNodeName="hasys03">
  <IPAddress baseAddress="19.126.124.32"
    subnetMask="255.255.255.0"
    networkName="db2_public_network_0"/>
</Interface>

<Interface interfaceName="eth0" clusterNodeName="hasys04">
  <IPAddress baseAddress="19.126.124.33"
    subnetMask="255.255.255.0"
    networkName="db2_public_network_0"/>
</Interface>

</PhysicalNetwork>

<!-- ===== -->
<!-- = Create a network named db2_private_network_0 with an IP      = -->
<!-- = network protocol.                                           = -->
<!-- = This network contains four computers: hasys01, hasys02,     = -->
<!-- = hasys03, and hasys04 (same as db2_public_network_0.)       = -->
<!-- = In addition to eth0, each computer has a network interface = -->
<!-- = card called eth1.                                           = -->
<!-- = The IP address of eth1 on hasys01 is 192.168.23.101        = -->
<!-- = The IP address of eth1 on hasys02 is 192.168.23.102        = -->
<!-- = The IP address of eth1 on hasys03 is 192.168.23.103        = -->
<!-- = The IP address of eth1 on hasys04 is 192.168.23.104        = -->
<!-- ===== -->
<PhysicalNetwork physicalNetworkName="db2_private_network_0"
  physicalNetworkProtocol="ip">

  <Interface interfaceName="eth1" clusterNodeName="hasys01">
    <IPAddress baseAddress="192.168.23.101"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

  <Interface interfaceName="eth1" clusterNodeName="hasys02">
    <IPAddress baseAddress="192.168.23.102"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

  <Interface interfaceName="eth1" clusterNodeName="hasys03">
    <IPAddress baseAddress="192.168.23.103"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

  <Interface interfaceName="eth1" clusterNodeName="hasys04">
    <IPAddress baseAddress="192.168.23.104"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

</PhysicalNetwork>

<!-- ===== -->
<!-- = List the computers (cluster nodes) in the cluster domain. = -->
<!-- ===== -->
<ClusterNode clusterNodeName="hasys01"/>

```

```

    <ClusterNode clusterNodeName="hasys02"/>
    <ClusterNode clusterNodeName="hasys03"/>
    <ClusterNode clusterNodeName="hasys04"/>

</ClusterDomain>

<!-- ===== -->
<!-- = The failover policy specifies the order in which the cluster = -->
<!-- = domain nodes should fail over. = -->
<!-- ===== -->
<FailoverPolicy>
    <Mutual />
</FailoverPolicy>

<!-- ===== -->
<!-- = Specify all the details of the database partitions. = -->
<!-- ===== -->
<DB2PartitionSet>

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
        <VirtualIPAddress baseAddress="19.126.124.251"
            subnetMask="255.255.255.0"
            networkName="db2_public_network_0"/>
        <Mount filesystemPath="/hafs/db2inst1/NODE0000"/>
        <MutualPair systemPairNode1="hasys01" systemPairNode2="hasys02" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="1" instanceName="db2inst1">
        <Mount filesystemPath="/hafs/db2inst1/NODE0001"/>
        <MutualPair systemPairNode1="hasys02" systemPairNode2="hasys01" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="2" instanceName="db2inst1">
        <Mount filesystemPath="/hafs/db2inst1/NODE0002"/>
        <MutualPair systemPairNode1="hasys03" systemPairNode2="hasys04" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="3" instanceName="db2inst1">
        <Mount filesystemPath="/hafs/db2inst1/NODE0003"/>
        <MutualPair systemPairNode1="hasys04" systemPairNode2="hasys03" />
    </DB2Partition>

</DB2PartitionSet>

<!-- ===== -->
<!-- = List of databases to be configured for High Availability = -->
<!-- ===== -->
<HADBSet instanceName="db2inst1">
    <HADB databaseName = "SAMPLE" />
    <HADB databaseName = "MYDB" />
</HADBSet>

</DB2Cluster>

```

*db2ha\_sample\_DPF\_NPlusM.xml:*

The sample file `db2ha_sample_DPF_NPlusM.xml` is an example of an XML input file that you pass to Db2 high availability instance configuration utility (**db2haicu**) to specify a new *cluster domain*. `db2ha_sample_DPF_NPlusM.xml` is located in the `sqlib/samples/ha/xml` directory.

## Features

The db2ha\_sample\_DPF\_NPlusM.xml sample demonstrates how to use **db2haicu** with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): four
- failover policy: N Plus M
- database partitions: two
- virtual (service) IP addresses: one
- shared mount points for failover: four

## XML source

```
<!-- ===== -->
<!-- = Use the Db2 High Availability Instance Configuration Utility = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP) = -->
<!-- = Base Component as the cluster manager. = -->
<!-- ===== -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="db2ha.xsd"
  clusterManagerName="TSA"
  version="1.0">

  <!-- ===== -->
  <!-- = Create a cluster domain named db2HADomain. = -->
  <!-- ===== -->
  <ClusterDomain domainName="db2HADomain">

    <!-- ===== -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5). = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes. = -->
    <!-- ===== -->
    <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

    <!-- ===== -->
    <!-- = Create a network named db2_public_network_0 with an IP = -->
    <!-- = network protocol. = -->
    <!-- = This network contains four computers: hasys01, hasys02, = -->
    <!-- = hasys03, and hasys04. = -->
    <!-- = Each computer has a network interface card called eth0. = -->
    <!-- = The IP address of eth0 on hasys01 is 19.126.124.30 = -->
    <!-- = The IP address of eth0 on hasys02 is 19.126.124.31 = -->
    <!-- = The IP address of eth0 on hasys03 is 19.126.124.32 = -->
    <!-- = The IP address of eth0 on hasys04 is 19.126.124.33 = -->
    <!-- ===== -->
    <PhysicalNetwork physicalNetworkName="db2_public_network_0"
      physicalNetworkProtocol="ip">

      <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.124.30"
          subnetMask="255.255.255.0"
          networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys02">
        <IPAddress baseAddress="19.126.124.31"
          subnetMask="255.255.255.0"
          networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys03">
        <IPAddress baseAddress="19.126.124.32"
```

```

        subnetMask="255.255.255.0"
        networkName="db2_public_network_0"/>
</Interface>

<Interface interfaceName="eth0" clusterNodeName="hasys04">
  <IPAddress baseAddress="19.126.124.33"
    subnetMask="255.255.255.0"
    networkName="db2_public_network_0"/>
</Interface>

</PhysicalNetwork>

<!-- ===== -->
<!-- = Create a network named db2_private_network_0 with an IP      = -->
<!-- = network protocol.                                           = -->
<!-- = This network contains four computers: hasys01, hasys02,     = -->
<!-- = hasys03, and hasys04 (same as db2_public_network_0.)       = -->
<!-- = In addition to eth0, each computer has a network interface = -->
<!-- = card called eth1.                                           = -->
<!-- = The IP address of eth1 on hasys01 is 192.168.23.101        = -->
<!-- = The IP address of eth1 on hasys02 is 192.168.23.102        = -->
<!-- = The IP address of eth1 on hasys03 is 192.168.23.103        = -->
<!-- = The IP address of eth1 on hasys04 is 192.168.23.104        = -->
<!-- ===== -->
<PhysicalNetwork physicalNetworkName="db2_private_network_0"
  physicalNetworkProtocol="ip">

  <Interface interfaceName="eth1" clusterNodeName="hasys01">
    <IPAddress baseAddress="192.168.23.101"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

  <Interface interfaceName="eth1" clusterNodeName="hasys02">
    <IPAddress baseAddress="192.168.23.102"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

  <Interface interfaceName="eth1" clusterNodeName="hasys03">
    <IPAddress baseAddress="192.168.23.103"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

  <Interface interfaceName="eth1" clusterNodeName="hasys04">
    <IPAddress baseAddress="192.168.23.104"
      subnetMask="255.255.255.0"
      networkName="db2_private_network_0"/>
  </Interface>

</PhysicalNetwork>

<!-- ===== -->
<!-- = List the computers (cluster nodes) in the cluster domain.  = -->
<!-- ===== -->
<ClusterNode clusterNodeName="hasys01"/>
<ClusterNode clusterNodeName="hasys02"/>
<ClusterNode clusterNodeName="hasys03"/>
<ClusterNode clusterNodeName="hasys04"/>

</ClusterDomain>

<!-- ===== -->
<!-- = The failover policy specifies the order in which the cluster = -->
<!-- = domain nodes should fail over.                               = -->

```

```

<!-- ===== -->
<FailoverPolicy>
  <NPlusM />
</FailoverPolicy>

<!-- ===== -->
<!-- = Specify all the details of the database partitions = -->
<!-- ===== -->
<DB2PartitionSet>

  <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
    <VirtualIPAddress baseAddress="19.126.124.250"
      subnetMask="255.255.255.0"
      networkName="db2_public_network_0"/>
    <Mount filesystemPath="/ha_dpfl/db2inst1/NODE0000"/>
    <Mount filesystemPath="/hafs/NODE0000"/>
    <NPlusMNode standbyNodeName="hasys03" />
  </DB2Partition>

  <DB2Partition dbpartitionnum="1" instanceName="db2inst1">
    <Mount filesystemPath="/ha_dpfl/db2inst1/NODE0001"/>
    <Mount filesystemPath="/hafs/NODE0001"/>
    <NPlusMNode standbyNodeName="hasys04" />
  </DB2Partition>

</DB2PartitionSet>

</DB2Cluster>

```

*db2ha\_sample\_HADR.xml:*

The sample file `db2ha_sample_DPF_HADR.xml` is an example of an XML input file that you pass to Db2 high availability instance configuration utility (**db2haicu**) to specify a new *cluster domain*. `db2ha_sample_HADR.xml` is located in the `sqlib/samples/ha/xml` directory.

## Features

The `db2ha_sample_HADR.xml` sample demonstrates how to use **db2haicu** with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): two
- failover policy: HADR
- database partitions: one
- virtual (service) IP addresses: none
- shared mount points for failover: none

## XML source

```

<!-- ===== -->
<!-- = Db2 High Availability configuration schema = -->
<!-- = Schema describes the elements of Db2 High Availability = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP) = -->
<!-- = that are used in the configuration of a HA cluster = -->
<!-- ===== -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="db2ha.xsd" cluster ManagerName="TSA" version="1.0">

  <!-- ===== -->
  <!-- = ClusterDomain element = -->
  <!-- = This element encapsulates the cluster configuration = -->

```



```

<!-- = specification = -->
<!-- = Creating cluster domain of name db2HAdomain = -->
<!-- = Creating an IP quorum device (IP 19.126.4.5) = -->
<!-- = The IP must be pingable at all times by each of the nodes in = -->
<!-- = the cluster domain = -->
<!-- ===== -->
<ClusterDomain domainName="db2HAdomain">
  <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

<!-- ===== -->
<!-- = Physical network element = -->
<!-- = The physical network specifies the network type, protocol = -->
<!-- = IP address, subnet mask, and NIC name = -->
<!-- = Define two logical groupings of NICs = -->
<!-- = Define two logical groupings of NICs = -->
<!-- ===== -->
<PhysicalNetwork physicalNetworkName="db2_public_network_0"
physicalNetworkProtocol="ip">
  <Interface interfaceName="eth0" clusterNodeName="hasys01">
    <IPAddress baseAddress="19.126.52.139"
subnetMask="255.255.255.0" networkName="db2_public_network_0"/>
  </Interface>
  <Interface interfaceName="eth0" clusterNodeName="hasys02">
    <IPAddress baseAddress="19.126.52.140"
subnetMask="255.255.255.0" networkName="db2_public_network_0"/>
  </Interface>
</PhysicalNetwork>

  <PhysicalNetwork physicalNetworkName="db2_private_network_0"
physicalNetworkProtocol="ip">
    <Interface interfaceName="eth1" clusterNodeName="hasys01">
      <IPAddress baseAddress="192.168.23.101"
subnetMask="255.255.255.0" networkName="db2_private_network_0"/>
    </Interface>
    <Interface interfaceName="eth1" clusterNodeName="hasys02">
      <IPAddress baseAddress="192.168.23.102"
subnetMask="255.255.255.0" networkName="db2_private_network_0"/>
    </Interface>
  </PhysicalNetwork>

<!-- ===== -->
<!-- = ClusterNodeName element = -->
<!-- = The set of nodes in the cluster domain = -->
<!-- = Here the defined set of nodes in the domain is = -->
<!-- = hasys01, hasys02 = -->
<!-- ===== -->
<ClusterNode clusterNodeName="hasys01"/>
<ClusterNode clusterNodeName="hasys02"/>
</ClusterDomain>

<!-- ===== -->
<!-- = Failover policy element = -->
<!-- = The failover policy specifies the failover order of the = -->
<!-- = cluster nodes = -->
<!-- = In the current sample the failover policy is to restart = -->
<!-- = instance in place (LocalRestart) = -->
<!-- ===== -->
<FailoverPolicy>
  <HADRFailover></HADRFailover>
</FailoverPolicy>

<!-- ===== -->
<!-- = Db2 Partition element = -->
<!-- = The Db2 partition type specifies a Db2 Instance Name, = -->
<!-- = partition number = -->
<!-- ===== -->
<DB2PartitionSet>

```

```

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
    </DB2Partition>
</DB2PartitionSet>

<!-- ===== -->
<!-- = HADRDBSet = -->
<!-- = Set of HADR Databases for this instance = -->
<!-- = Specify the databaseName, the name of the local instance on = -->
<!-- = this machine controlling the HADR database, the name of the = -->
<!-- = remote instance in this HADR pair, the name of the local = -->
<!-- = hostname and the remote hostname for the remote instance = -->
<!-- ===== -->
<HADRDBSet>
  <HADRDB databaseName="HADRDB" localInstance="db2inst1"
    remoteInstance="db2inst1" localHost="hasys01" remoteHost="hasys02"/>
</HADRDBSet>
</DB2Cluster>

```

### Db2 High Availability Instance Configuration Utility (db2haicu) prerequisites:

There is a set of tasks you must perform before using Db2 high availability instance configuration utility (**db2haicu**).

#### General

Before a database manager instance owner can run **db2haicu**, a user with root authority must run the **preprnode** command. **preprnode** is part of the Reliable Scalable Cluster Technology (RSCT) fileset for AIX and the RSCT package for Linux. **preprnode** handles initializing the nodes for intracluster communication. The **preprnode** command is run as a part of setting up the cluster. For more information about **preprnode**, see: **preprnode** Command. For more information about RSCT, see RSCT Administration Guide - What is RSCT?

Also, a user with root authority must disable the **iTCO\_wdt** and **iTCO\_vendor\_support** modules.

- On SUSE, add the following lines to the **/etc/modprobe.d/blacklist** file:

```
alias iTCO_wdt off
alias iTCO_vendor_support off
```
- On RHEL 5.x, add the following lines to the **/etc/modprobe.conf** file:

```
blacklist iTCO_wdt
blacklist iTCO_vendor_support
```
- On RHEL 6 and later releases, create the **/etc/modprobe.d/iTCO\_wdt.conf** and **/etc/modprobe.d/iTCO\_vendor\_support.conf** files as follows:

```
[root@host1]# cat /etc/modprobe.d/iTCO_wdt.conf
blacklist iTCO_wdt
[root@host1]# cat /etc/modprobe.d/iTCO_vendor_support.conf
blacklist iTCO_vendor_support
```

You can verify that the modules are disabled by using the **lsmod** command.

Before running **db2haicu**, a database manager instance owner must perform the following tasks:

- Synchronize services files on all machines that are being added to the cluster.
- Run the **db2profile** script for the database manager instance that is being used to create the cluster domain.
- Start the database manager using the **db2start** command.

For Linux, the GNU C library (glibc) version must be 2.4-31.109.1 or higher.

## Db2 high availability disaster recovery (HADR)

If you will be using HADR functionality, perform the following tasks:

- 
- Ensure that both HADR databases exist on different systems.
- Ensure all HADR databases are started in their respective primary and standby database roles, and that all HADR primary-standby database pairs are in peer state.
- Ensure that you are using one of the following HADR synchronization modes: SYNC or NEARSYNC.
- Configure **hadr\_peer\_window** for all HADR databases to a value of at least 120 seconds.
- Disable the Db2 fault monitor.

## Partitioned database environment

If you have multiple database partitions to configure for high availability, perform the following steps:

- Configure the **DB2\_NUM\_FAILOVER\_NODES** registry variable on all machines that are being added to the cluster domain.
- (Optional) Activate the database before running **db2haicu**.

## Creating a cluster domain using Db2 High Availability Instance Configuration Utility (db2haicu):

When you run Db2 high availability instance configuration utility (**db2haicu**) for the first time for a database manager instance, **db2haicu** creates a model of your cluster, called a *cluster domain*.

*Database paths detected automatically by Db2 High Availability Instance Configuration Utility (db2haicu):*

When you run Db2 high availability instance configuration utility (**db2haicu**) for the first time, **db2haicu** will search your database system for database configuration information that is related to cluster configuration.

## Single database partition environment

In a single database partition environment, **db2haicu** automatically detects the paths:

- Instance home directory path
- Audit log path
- Audit archive log path
- Sync point manager (SPM) log path
- Db2 diagnostic log (**db2diag** log file) path
- Database related paths:
  - Database log path
  - Database table space container path
  - Database table space directory path
  - Local database directory

**Note:** If any of the database related directories are symbolic links, the **db2haicu** utility prints an error message and exits.

### Multiple database partition environment

In a multiple database partition environment, **db2haicu** automatically detects only the following paths:

- Database log path
- Database table space container path
- Database table space directory path
- Local database directory

### Maintaining a cluster domain using Db2 High Availability Instance Configuration Utility (db2haicu):

When you are modifying the cluster domain model of your clustered environment using **db2haicu**, the database manager propagates the related changes to your database manager instance and cluster configuration.

### Before you begin

Before you can configure your clustered environment using **db2haicu**, you must create and configure a cluster domain. For more information, see “Creating a cluster domain using Db2 High Availability Instance Configuration Utility (db2haicu)” on page 87

### About this task

**db2haicu** maintenance tasks include adding cluster elements such as databases or cluster nodes to the cluster domain, and removing elements from the cluster domain. **db2haicu** maintenance tasks also include modifying the details of cluster domain elements such as the failover policy for the database manager instance.

### Procedure

#### 1. Run **db2haicu**

When you run **db2haicu** in maintenance mode, **db2haicu** presents you with a list of operations you can perform on the cluster domain:

- Add or remove cluster nodes (machine identified by hostname)
- Add or remove a network interface (network interface card)
- Add or remove database partitions (partitioned database environment only)
- Add or remove a Db2 High Availability Disaster Recovery (HADR) database
- Add or remove a highly available database
- Add or remove a mount point
- Add or remove an IP address
- Add or remove a non-critical path
- Move database partitions and HADR databases for scheduled maintenance
- Change failover policy for the current instance
- Create a new quorum device for the cluster domain
- Destroy the cluster domain

#### 2. Select a task to perform, and answer subsequent questions that **db2haicu** presents.

## Results

The database manager uses the information in the cluster domain to coordinate with your cluster manager. When you configure your database and cluster elements using **db2haicu** then those elements are included in integrated and automated cluster configuration and administration provided by the Db2 High Availability (HA) Feature. When you use **db2haicu** to make a database manager instance configuration change, the database manager makes the required cluster manager configuration change for you so that you do not have to make a subsequent call to your cluster manager.

## What to do next

Db2 high availability instance configuration utility (**db2haicu**) does not have a separate diagnostic log. You can investigate and diagnose **db2haicu** errors using the database manager diagnostic log, **db2diag** log file, and the **db2pd** tool. For more information, see: "Troubleshooting Db2 High Availability Instance Configuration Utility (db2haicu)"

## Troubleshooting Db2 High Availability Instance Configuration Utility (db2haicu):

Db2 high availability instance configuration utility (**db2haicu**) does not have a separate diagnostic log. You can investigate and diagnose **db2haicu** errors using the database manager diagnostic log, **db2diag** log file, and the **db2pd** tool.

## Db2 high availability instance configuration utility (db2haicu) restrictions:

There are some restrictions for using the Db2 high availability instance configuration utility (**db2haicu**).

- "Software and hardware"
- "Configuration tasks"
- "Usage notes"
- "Recommendations" on page 91

## Software and hardware

- **db2haicu** does not support the configuration of mount resources which are based off of Logical Volume Manager (LVM) on any platform other than AIX.

## Configuration tasks

You cannot perform the following tasks using **db2haicu**:

- You cannot configure automatic client reroute using **db2haicu**.
- When you upgrade from Db2 Version 10.5 to a later version, you cannot use **db2haicu** to migrate your cluster configuration. To migrate a cluster configuration, you must perform the following steps:
  1. Delete the existing cluster domain (if one exists)
  2. Upgrade the database server
  3. Create a new cluster domain using **db2haicu**

## Usage notes

**db2haicu** is not supported in a Db2 pureScale environment. Use the **db2cluster** command instead to configure clustered environments.

**Important:** For an HADR configuration, **db2haicu** only creates Db2 resource dependencies against network equivalencies that are defined by **db2\_public\_network\_\***.

Consider the following **db2haicu** usage notes when planning your cluster configuration and administration activities:

- Even though **db2haicu** performs some administration tasks that normally require root authority, **db2haicu** runs with the privileges of the database manager instance owner. **db2haicu** initialization, which is performed by a root user, enables **db2haicu** to carry out the required configuration changes despite having only instance owner privileges.
- When you create a new cluster domain, **db2haicu** does not verify that the name you specify for the new cluster domain is valid. For example, **db2haicu** does not confirm that the name is a valid length, or contains valid characters, or that is not the same name as an existing cluster domain.
- **db2haicu** does not verify or validate information that a user specifies and that is passed to a cluster manager. Because **db2haicu** cannot be aware of all cluster manager restrictions with respect to cluster object names, for example, **db2haicu** passes text to the cluster manager without validating it for things like valid characters, or length.
- If an error happens and **db2haicu** fails while you are creating and configuring a new cluster domain, you must perform the following steps:
  1. Remove the resource groups of the partially created cluster domain by running **db2haicu** using the **-delete** parameter
  2. Re-create the new cluster domain by calling **db2haicu** again.
- When you run **db2haicu** with the **-delete** parameter, **db2haicu** deletes the resource groups that are associated with the current database manager instance immediately, without confirming whether those resource groups are locked.
- To remove resource groups that are associated with the database manager instances of a Db2 high availability disaster recovery (HADR) database pair, perform the following steps:
  1. Run **db2haicu** with the **-delete** parameter against the database manager instance of the HADR standby database first.
  2. Run **db2haicu** with the **-delete** parameter against the database manager instance of the HADR primary database.
- To remove a virtual IP from an HADR resource group using **db2haicu**, you must remove it from the instance on which it was created.
- The ASYNC and SUPERASYNC HADR synchronization modes are not supported by **db2haicu**.
- If a cluster operation you attempt to perform using **db2haicu** times out, **db2haicu** does not return an error to you. When a cluster operation times out, you do not know that the operation timed out unless you review diagnostic logs after you make the **db2haicu** call; or unless a subsequent cluster action fails, and while you are investigating that subsequent failure, you determine that the original cluster operation timed out.
- If you attempt to change the failover policy for a given database instance to active-passive, there is one condition under which that configuration operation fails, but for which **db2haicu** does not return an error to you. If you specify a machine that is currently offline to be the *active* machine, **db2haicu** does not make that machine the active machine, but **db2haicu** does not return an error that indicates that the change did not succeed.

- For a shared disk configuration, **db2haicu** does not support a nested mount configuration because Db2 does not enforce the disk mount order.
- When you are adding network interface cards (NICs) to a network, you cannot add NICs with different subnet masks to the same network using **db2haicu**. If you want to add NICs with different subnet masks to the same network, use the following SA MP command:  

```
mkequ <name> IBM.NetworkInterface:<eth0>:<node0>,...,<ethN>:<nodeN>
```
- For a shared disk configuration, file systems based on Concurrent Enhanced-Capable volume groups are not supported by **db2haicu**.

## Recommendations

The following is a list of recommendations for configuration your cluster, and your database manager instances when you are using **db2haicu**.

- When you add new mount points for the cluster by adding entries to `/etc/fstab`, use the **noauto** option to prevent the mount points from being automatically mounted on more than one machine in the cluster. For example:  

```
dev/vpath1      /db/svtpdb/NODE0010      ext3  noauto 0 0
```

## High availability through log shipping

Log shipping is the process of copying whole log files to a standby machine either from an archive device, or through a user exit program that is running against the primary database. A scheduled job on the standby issues the **ROLLFORWARD DATABASE** command at a specified interval to keep the standby current in terms of log replay.

To set up log shipping, you configure the database with user exit programs and log archiving enabled, initialize the standby, modify one of the sample user exit files (you can find more information about them in “Log management user exit samples”) to either archive the logs to a shared device or send the log files to the standby’s log path, and schedule a job so that the standby processes the log files. For a detailed overview of setting up log shipping, see the following article: [article](#).

After your log shipping solution is configured, the standby database is continuously rolling forward through the log files that are produced by the production machine. When the production machine fails, a failover occurs (you either initiate the failover manually or create a script to do this) and the following takes place:

- The remaining logs are transferred over to the standby machine.
- The standby database rolls forward to the end of the logs and stops.
- The clients reconnect to the standby database, which is now the new primary, and resume operations.

The standby machine has its own resources (for example, disks), but must have the same physical and logical definitions as the production database. When using this approach, create the initial standby database by using restore utility (from a backup of the primary database) or by using the split mirror function if that is available.

To ensure that you are able to recover your database in a disaster recovery situation, consider the following recommendations:

- Ensure that the archive location is geographically separate from the primary site.
- Remotely mirror the logs at the standby database site.

- Use a synchronous mirror for no-loss support. You can do this using modern disk subsystems such as ESS and EMC, or another remote mirroring technology. NVRAM cache (both local and remote) is also recommended to minimize the performance impact of a disaster recovery situation.

If you want to control which log files are to be rolled forward on the standby machine, you can disable the retrieval of archived logs by using the **NORETRIEVE** option with the **ROLLFORWARD DATABASE** command. The benefits of this option are as follows:

- By controlling the log files to be rolled forward, you can ensure that the standby machine is X hours behind the production machine, which avoids affecting both the systems.
- If the standby system does not have access to archive (for example, if Tivoli Storage Manager is the archive, it allows only the original machine to retrieve the files).
- It might also be possible that while the production system is archiving a file, the standby system is retrieving the same file, and it might then get an incomplete log file. The **NORETRIEVE** option solves this problem.

**Note:**

1. When the standby database processes a log record which indicates that an index rebuild took place on the primary database, the indexes on the standby server are not automatically rebuilt. The index is rebuilt on the standby server either at the first connection to the database, or at the first attempt to access the index after the standby server is taken out of the rollforward pending state. It is recommended that the standby server be resynchronized with the primary server if any indexes on the primary server are rebuilt. You can enable indexes to be rebuilt during rollforward operations if you set the **logindexbuild** database configuration parameter.
2. If the load utility is run on the primary database with the **COPY YES** option specified, the standby database must have access to the copy image.
3. If the load utility is run on the primary database with the **COPY NO** option specified, the standby database must be resynchronized or the table space is placed in restore pending state.
4. There are two ways to initialize a standby machine:
  - a. By restoring to it from a backup image.
  - b. By creating a split mirror of the production system and issuing the **db2inidb** command with the **STANDBY** option.

Only after the standby machine is initialized can you issue the **ROLLFORWARD DATABASE** command on the standby system.

5. Operations that are not logged are not replayed on the standby database. As a result, it is recommended that you resynchronize the standby database after such operations. You can do this resynchronization through online split mirror and suspended I/O support.

## Log mirroring

IBM Db2 server supports log mirroring at the database level. Mirroring log files helps protect a database from accidental deletion of an active log and data corruption caused by hardware failure.



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

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

When you first give a value to the **mirrorlogpath** configuration parameter, Db2 will not use it until the next database startup. This behavior is similar to the **newlogpath** configuration parameter.

If there is an error writing to either the active log path or the mirror log path, the database marks the failing path as “bad”, writes a message to the administration notification log, and writes subsequent log records only to the remaining “good” log path. Db2 does not attempt to use the “bad” path again until the current log file is either full or truncated. When Db2 needs to open the next log file, it verifies that this path is valid, and if so, begins to use it. If not, Db2 does not attempt to use the path again until the next log file is accessed for the first time. There is no attempt to synchronize the log paths, but Db2 keeps information about access errors that occur, so that the correct paths are used when log files are archived. If a failure occurs while writing to the remaining “good” path, the database shuts down.

## High availability through suspended I/O and online split mirror support

IBM Db2 server suspended I/O support enables you to split mirrored copies of your primary database without taking the database offline. You can use this to very quickly create a standby database to take over if the primary database fails.

Disk mirroring is the process of writing data to two separate hard disks at the same time. One copy of the data is called a mirror of the other. Splitting a mirror is the process of separating the two copies.

You can use disk mirroring to maintain a secondary copy of your primary database. You can use Db2 server suspended I/O functionality to split the primary and secondary mirrored copies of the database without taking the database offline. Once the primary and secondary databases copies are split, the secondary database can take over operations if the primary database fails.

If you would rather not back up a large database using the Db2 server backup utility, you can make copies from a mirrored image by using suspended I/O and the split mirror function. This approach also:

- Eliminates backup operation overhead from the production machine
- Represents a fast way to clone systems
- Represents a fast implementation of idle standby failover. There is no initial restore operation, and if a rollforward operation proves to be too slow, or encounters errors, reinitialization is very fast.

The **db2inidb** command initializes the split mirror so that it can be used:

- As a clone database

- As a standby database
- As a backup image

This command can only be issued against a split mirror, and it must be run before the split mirror can be used.

In a partitioned database environment, you do not have to suspend I/O writes on all database partitions simultaneously. You can suspend a subset of one or more database partitions to create split mirrors for performing offline backups. If the catalog partition is included in the subset, it must be the last database partition to be suspended.

In a partitioned database environment, the **db2inidb** command must be run on every database partition before the split image from any of the database partitions can be used. The tool can be run on all database partitions simultaneously using the **db2\_all** command. If; however, you are using the RELOCATE USING option, you cannot use the **db2\_all** command to run **db2inidb** on all of the database partitions simultaneously. A separate configuration file must be supplied for each database partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the **db2relocatedb** command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the **db2relocatedb** command only needs to be run once on that database partition.

**Note:** Ensure that the split mirror contains all containers and directories which comprise the database, including the volume directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.

### Using a split mirror as a standby database

Use the following procedure to create a split mirror of a database for use as a standby database outside of a Db2 pureScale environment.

If a failure occurs on the primary database and it becomes inaccessible, you can use the standby database to take over for the primary database.

### About this task

If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it during rollforward operations. However, once the database is brought out of the rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. While the standby database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the standby database. This could cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa , and can affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid recoverability issues.

## Procedure

To use a split mirror as a standby database:

1. Connect to the primary database using the following command:  
`db2 connect to db_name`
2. Suspend the I/O write operations on the primary database using the following command:  
`db2 set write suspend for database`

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally use the `FLUSH BUFFERPOOLS ALL` statement before issuing **SET WRITE SUSPEND** to minimize the recovery time of the standby database.

3. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

**Note:**

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the `DBPATHS` administrative view, which shows all the files and directories of the database that need to be split.
  - If you specified the `EXCLUDE LOGS` with the **SET WRITE** command, do not include the log files in the copy.
4. Resume the I/O write operations on the primary database using the following command:  
`db2 set write resume for database`
  5. Catalog the mirrored database on the secondary system.

**Note:** By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the `RELOCATE USING` option of the **db2inidb** command to accomplish this.

6. Start the database instance on the secondary system using the following command:  
`db2start`
7. Initialize the mirrored database on the secondary system by placing it in rollforward pending state using the following command:  
`db2inidb <database_alias> as standby`

If required, specify the `RELOCATE USING` option of the **db2inidb** command to relocate the standby database:

```
db2inidb <database_alias> as standby relocate using relocatedbcfg.txt
```

where the `relocatedbcfg.txt` file contains the information required to relocate the database.

**Note:** You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror reduces the overhead of taking a backup on the production database. Such backups are considered to be online

backups and will contain in-flight transactions, but you cannot include log files from the standby database. When such a backup is restored, you must rollforward to at least the end of the backup before you can issue a **ROLLFORWARD** command with the **STOP** option. Because the backup will not contain any log files, the log files from the primary database that were in use at the time the **SET WRITE SUSPEND** command was issued must be available or the rollforward operation will not be able to reach the end of the backup.

8. Make the archived log files from the primary database available to the standby database either by configuring the log archiving parameters on the standby database or by shipping logs to the standby database.
9. Rollforward the database to the end of the logs or to a point-in-time.
10. Continue retrieving log files and rollforwarding the database through the logs until you reach the end of the logs or the point-in-time required for the standby database.
11. Bring the standby database online by issuing the **ROLLFORWARD** command with the **STOP** option specified.

**Note:**

- The logs from the primary database cannot be applied to the mirrored database after it has been taken out of rollforward pending state.
- If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it while rollforward is being performed. However, once the database is brought out of rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. Although the standby database will initially use a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the standby database. This may cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa. This could affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid these issues.

## **Using a split mirror as a standby database in a Db2 pureScale environment**

Use the following procedure to create a split mirror of a database for use as a standby database in a Db2 pureScale environment. If a failure occurs on the primary database and it becomes inaccessible, you can use the standby database to take over for the primary database.

### **About this task**

If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it during rollforward operations. However, once the database is brought out of the rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. While the standby database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the standby database. This could cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa, and can affect the recoverability of both databases. You should change the log archiving destination

for the standby database to be different from that of the primary database to avoid recoverability issues.

## Procedure

To use a split mirror as a standby database:

1. Connect to the primary database using the following command:  
`db2 connect to <db_name>`
2. Configure the General Parallel File System (GPFS™) on the secondary cluster by extracting and importing the primary cluster's settings. On the primary cluster, run the following GPFS command:  
`mmfsctl <filesystem> syncFSconfig -n <remotenodefile>`

where *<remotenodefile>* is the list of hosts in the secondary cluster.

3. List the cluster manager domain using the following command:  
`db2cluster -cm -list -domain`
4. Stop the cluster manager on each host in the cluster using the following command:  
`db2cluster -cm -stop -host <host> -force`

**Note:** The last host which you shut down must be the host from which you are issuing this command.

5. Stop the GPFS cluster on the secondary system using the following command:  
`db2cluster -cfs -stop -all`
6. Suspend the I/O write operations on the primary database using the following command:  
`db2 set write suspend for database`

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the `FLUSH BUFFERPOOLS ALL` statement.

7. Determine which file systems must be suspended and copied using the following command:  
`db2cluster -cfs -list -filesystem`
8. Suspend each GPFS file system that contains data or log data using the following command:  
`/usr/lpp/mmfs/bin/mmfsctl <filesystem> suspend`

where *<filesystem>* represents a file system that contains data or log data.

**Note:** While the GPFS file systems are suspended, both read and write operations are blocked. You should only be performing the split mirror operations during this period to minimize the amount of time that read operations are blocked.

9. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

### Note:

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container

directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.

- If you specified the EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.

10. Resume the GPFS file systems that were suspended using the following command for each suspended file system:

```
/usr/lpp/mmfs/bin/mmfsctl <filesystem> resume
```

where *filesystem* represents a suspended file system that contains data or log data.

11. Resume the I/O write operations on the primary database using the following command:

```
db2 set write resume for database
```

12. Start the GPFS cluster on the secondary system using the following command:

```
db2cluster -cfs -start -all
```

13. Start the cluster manager using the following command

```
db2cluster -cm -start -domain <domain>
```

14. Catalog the mirrored database on the secondary system.

**Note:** By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the **RELOCATE USING** option of the **db2inidb** command to accomplish this.

15. Start the database instance on the secondary system using the following command:

```
db2start
```

16. Initialize the database on the secondary system by placing it in rollforward pending state:

```
db2inidb <database_alias> as standby
```

If required, specify the RELOCATE USING option of the **db2inidb** command to relocate the database:

```
db2inidb database_alias as standby relocate using relocatedbcfg.txt
```

where *relocatedbcfg.txt* contains the information required to relocate the database.

**Note:** You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror reduces the overhead of taking a backup on the production database. Such backups are considered to be online backups and will contain in-flight transactions, but you cannot include log files from the standby database. When such a backup is restored, you must rollforward to at least the end of the backup before you can issue a **ROLLFORWARD STOP** command. Because the backup will not contain any log files, the log files from the primary database that were in use at the time the **SET WRITE SUSPEND** command was issued must be available or the rollforward operation will not be able to reach the end of the backup.

17. Make the archived log files from the primary database available to the standby database either by configuring the log archiving parameters on the standby database or by shipping logs to the standby database.
18. Rollforward the database to the end of the logs or to a point-in-time.

**Note:** When executing rollforward operations, you might encounter SQL1273 errors. These errors are expected if some of the log files were not copied from the primary system when the database was split or if one member generates log files faster than other members. SQL1273 is generated in some cases when the rollforward operation must stop to preserve data consistency because the contents of the log files depends on the contents of unavailable log files from other members. If the standby database is configured to retrieve log files archived by the primary database, you can either wait for the primary system to archive the necessary log file or you can use the **ARCHIVE LOG** command on the primary system to force the log file to be archived. Otherwise, you must ship the required log files to the standby database. After the necessary log file is available on the standby database, the rollforward operation can read further ahead in the logs, although SQL1273 might be encountered again if some members are still generating log files faster than other members. For more information, see the “Disaster recovery and high availability through log shipping in a Db2 pureScale environment” section of the “Backup and restore operations in a Db2 pureScale environment” Information Center topic.

19. Continue the rollforward operation through the logs until you reach the end of the logs or the point-in-time required for the standby database, shipping new log files to the standby database if required.
20. Bring the standby database online by issuing the **ROLLFORWARD DATABASE** command with the **STOP** option specified.

**Note:**

- The logs from the primary database cannot be applied to the mirrored database once it has been taken out of rollforward pending state.
- If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it while rollforward is being performed. However, once the database is brought out of rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. Although the standby database will initially use a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the standby database. This may cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa. This could affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid these issues.

## Using a split mirror to clone a database

Use the following procedure to create a clone database in an environment outside of a Db2 pureScale environment. Although you can write to clone databases, they are generally used for read-only activities such as running reports.

## About this task

If the primary database was configured for log archiving, the cloned database will share the same log archiving configuration. If the archive log location is accessible to the cloned database, this could cause the cloned database to archive log files to the same location as the primary database and can affect the recoverability of both databases. While the cloned database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the cloned database. You should change the log archiving destination for the cloned database to be different from that of the primary database before running the **db2inidb** command to avoid recoverability issues.

You cannot back up a cloned database, restore the backup image on the original system, or roll forward through log files produced on the original system. The cloned database provides an instantaneous copy of the database only at that time when the I/O is suspended; any other outstanding uncommitted work will be rolled back after the **db2inidb** command is executed on the clone.

## Procedure

To clone a database:

1. Connect to the primary database using the following command:  
`db2 connect to db_name`
2. Suspend the I/O write operations on the primary database using the following command:  
`db2 set write suspend for database`

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the **FLUSH BUFFERPOOLS ALL** statement.

3. Create one or multiple split mirrors from the primary database using the appropriate operating system-level and storage-level commands.

### Note:

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
  - If you specified the **EXCLUDE LOGS** with the **SET WRITE** command, do not include the log files in the copy.
4. Resume the I/O write operations on the primary database using the following command:  
`db2 set write resume for database`
  5. Catalog the mirrored database on the secondary system.

**Note:** By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the **RELOCATE USING** option of the **db2inidb** command to accomplish this.



6. Start the database instance on the secondary system using the following command:

```
db2start
```

7. Initialize the mirrored database on the secondary system:

```
db2inidb database_alias as snapshot
```

If required, specify the RELOCATE USING option of the **db2inidb** command to relocate the clone database:

```
db2inidb database_alias as snapshot relocate using relocatedbcfg.txt
```

where the `relocatedbcfg.txt` file contains the information required to relocate the database.

**Note:**

- This command rolls back transactions that are in flight when the split occurs, and starts a new log chain sequence so that any logs from the primary database cannot be replayed on the cloned database.
- If the primary database was configured for log archiving, the cloned database will share the same log archiving configuration. This means that the cloned database attempts to archive log files to the same location used by the primary database if that location is accessible to the cloned database. Although the cloned database initially uses a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the cloned database. This might cause the primary database to archive log files on top of the log files archived by the clone database, or vice versa. This might affect the recoverability of both databases. You should change the log archiving destination for the cloned database to be different from that of the primary database to avoid these issues.

## Using a split mirror to clone a database in a Db2 pureScale environment

Use the following procedure to create a clone database in a Db2 pureScale environment. Although you can write to clone databases, they are generally used for read-only activities such as running reports.

### About this task

If the primary database was configured for log archiving, the cloned database will share the same log archiving configuration. If the archive log location is accessible to the cloned database, this could cause the cloned database to archive log files to the same location as the primary database and can affect the recoverability of both databases. While the cloned database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the cloned database. You should change the log archiving destination for the cloned database to be different from that of the primary database before running the **db2inidb** command to avoid recoverability issues.

You cannot back up a cloned database, restore the backup image on the original system, or roll forward through log files produced on the original system. The cloned database provides an instantaneous copy of the database only at that time when the I/O is suspended; any other outstanding uncommitted work will be rolled back after the **db2inidb** command is executed on the clone.

## Procedure

To clone a database:

1. Connect to the primary database using the following command:
2. Configure the General Parallel File System (GPFS) on the secondary cluster by extracting and importing the settings of the primary cluster. On the primary cluster, run the following GPFS command:

```
mmfsctl filesystem syncFSconfig -n remotenodefile
```

where *remotenodefile* is the list of hosts in the secondary cluster.

3. List the cluster manager domain using the following command:
4. Stop the cluster manager on each host in the cluster using the following command:

```
db2cluster -cm -list -domain
```

```
db2cluster -cm -stop -host host -force
```

**Note:** The last host which you shut down must be the host from which you are issuing this command.

5. Stop the GPFS cluster on the secondary system using the following command:

```
db2cluster -cfs -stop -all
```

6. Suspend the I/O write operations on the primary database using the following command:

```
db2 set write suspend for database
```

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the **FLUSH BUFFERPOOLS ALL** statement.

7. Determine which file systems must be suspended and copied using the following command:

```
db2cluster -cfs -list -filesystem
```

8. Suspend each GPFS file system that contains data or log data using the following command:

```
/usr/lpp/mmfs/bin/mmfsctl filesystem suspend-write
```

where *filesystem* represents a file system that contains data or log data.

**Note:** When the GPFS file systems are suspended, only write operations are blocked.

9. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

**Note:**

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
- If you specified the **EXCLUDE LOGS** with the **SET WRITE** command, do not include the log files in the copy.

10. Resume the GPFS file systems that were suspended using the following command for each suspended file system:

```
/usr/lpp/mmfs/bin/mmfsctl filesystem resume
```

where *filesystem* represents a suspended file system that contains data or log data.

11. Resume the I/O write operations on the primary database:  
`db2 set write resume for database`
12. Start the GPFS cluster on the secondary system using the following command:  
`db2cluster -cfs -start -all`
13. Start the cluster manager using the following command  
`db2cluster -cm -start -domain domain`
14. Catalog the mirrored database on the secondary system:

**Note:** By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the **RELOCATE USING** option of the **db2inidb** command to accomplish this.

15. Start the database instance on the secondary system using the following command:

```
db2start
```

16. Initialize the mirrored database on the secondary system using the following command:

```
db2inidb database_alias as snapshot
```

If required, specify the **RELOCATE USING** option of the **db2inidb** command to relocate the clone database:

```
db2inidb database_alias as snapshot relocate using relocatedbcfg.txt
```

where the `relocatedbcfg.txt` file contains the information required to relocate the database.

**Note:**

- This command rolls back transactions that are in flight when the split occurs, and starts a new log chain sequence so that any logs from the primary database cannot be replayed on the cloned database.
- If the primary database was configured for log archiving, the clone database shares the same log archiving configuration. If the log archiving destination is accessible to the cloned database, the standby database automatically retrieves log files from it while rollforward is being performed. However, once the database is brought out of rollforward pending state, the clone database attempts to archive log files to the same location used by the primary database. Although the standby database initially uses a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the cloned database. This might cause the primary database to archive log files on top of the log files archived by the cloned database, or vice versa. This might affect the recoverability of both databases. You should change the log archiving destination for the cloned database to be different from that of the primary database to avoid these issues.

## Using a split mirror as a backup image

Use the following procedure to create a split mirror of a database in a different location on the same system for use as a backup image outside of a Db2 pureScale environment. This procedure can be used instead of performing backup database operations on the database.

### Procedure

To use a split mirror as a backup image:

1. Connect to the primary database using the following command:  
`db2 connect to <db_name>`
2. Suspend the I/O write operations for the primary database using the following command:  
`db2 set write suspend for database`

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the **FLUSH BUFFERPOOLS ALL** statement.

3. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

#### Note:

- Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. If you are using multiple storage groups, you must copy all paths, including files and subdirectories of those paths. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
  - If you specified the **EXCLUDE LOGS** with the **SET WRITE** command, do not include the log files in the copy.
4. Resume the I/O write operations on the primary database using the following command:  
`db2 set write resume for database`

Assuming that a failure would occur on the system, perform the following steps to restore the database using the split-mirror database as the backup:

1. Stop the database instance using the following command:  
`db2stop`
2. Copy the split-off data using operating system-level commands.

**Important:** Do not copy the split-off log files, because the original logs are needed for rollforward recovery.

3. Start the database instance using the following command:  
`db2start`
4. Initialize the primary database:  
`db2inidb database_alias as mirror`

where *database\_alias* represents the database alias.

5. Rollforward the database to the end of the logs, or to a point-in-time, and stop.

## Using a split mirror as a backup image in a Db2 pureScale environment

Use the following procedure to create a split mirror of a database in a different location on the same system for use as a backup image in a Db2 pureScale environment. This procedure can be used instead of performing backup database operations on the database.

### Procedure

To use a split mirror as a backup image:

1. Connect to the primary database using the following command:  
`db2 connect to <db_name>`
2. Configure the General Parallel File System (GPFS) on the secondary cluster by extracting and importing the settings from the primary cluster. On the primary cluster, run the following GPFS command:  
`mmfsctl filesystem syncFSconfig -n remotenodefile`

where *remotenodefile* is the list of hosts in the secondary cluster.

3. Suspend the I/O write operations for the primary database using the following command:  
`db2 set write suspend for database`

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the **FLUSH BUFFERPOOLS ALL** statement.

4. Determine which file systems must be suspended and copied using the following command:  
`db2cluster -cfs -list -filesystem`
5. Suspend each GPFS file system that contains container data or log data using the following command:  
`/usr/lpp/mmfs/bin/mmfsctl filesystem suspend-write`

where *filesystem* represents a file system that contains data or log data.

**Note:** While the GPFS file systems are suspended, write operations are blocked. You should only be performing the split mirror operations during this period to minimize the amount of time that operations are blocked.

6. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

#### Note:

- Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. If you are using multiple storage groups, you must copy all paths, including files and subdirectories of those paths. To gather this information, refer to the **DBPATHS** administrative view, which shows all the files and directories of the database that need to be split.
  - If you specified the **EXCLUDE LOGS** with the **SET WRITE** command, do not include the log files in the copy.
7. Resume the GPFS file systems that were suspended using the following command for each suspended file system:

```
/usr/lpp/mmfs/bin/mmfsctl filesystem resume
```

where *filesystem* represents a suspended file system that contains data or log data.

8. Resume the I/O write operations for the primary database using the following command:

```
db2 set write resume for database
```

Assuming that a situation requires you to restore the database using the split mirror as the backup image, perform the following steps:

1. Stop the primary database instance using the following command:

```
db2stop
```

2. List the cluster manager domain using the following command:

```
db2cluster -cm -list -domain
```

3. Stop the cluster manager on each host in the cluster using the following command:

```
db2cluster -cm -stop -host host -force
```

**Note:** The last host which you shut down must be the host from which you are issuing this command.

4. Stop the GPFS cluster on the primary database instance using the following command:

```
db2cluster -cfs -stop -all
```

5. Copy the split-off data off the primary database using appropriate operating system-level commands.

**Important:** Do not copy the split-off log files, because the original logs are needed for rollforward recovery.

6. Start the GPFS cluster on the primary database instance using the following command:

```
db2cluster -cfs -start -all
```

7. Start the cluster manager using the following command

```
db2cluster -cm -start -domain domain
```

8. Start the database instance using the following command:

```
db2start
```

9. Initialize the primary database using the following command:

```
db2inidb database_alias as mirror
```

10. Rollforward the primary database to the end of the logs, or to a point-in-time, and stop.

---

## Configuring for high availability

To configure your Db2 database solution for high availability, you must: schedule database maintenance activities; configure the primary and standby database servers to know about each other and their respective roles in the event of a failure; and configure any cluster managing software to transfer workload from a failed cluster node.

### Before you begin

Before configuring your database solution:

- Assemble and install the underlying hardware and software components that make up the solution. These underlying components might include: power supply; network connectivity; network cards; disks or other storage devices; operating systems; and cluster managing software.
- Test these underlying components without any database workload to make sure they are functioning properly before attempting to use them in database load balancing, failover, or recovery operations.

### About this task

Redundancy is an important part of a high availability solution. However, if you do not schedule maintenance wisely, if you run out of storage space for needed recovery logs, or if your cluster managing software is not configured correctly, your solution might not be available when your users need to do crucial work with the database.

### Procedure

- Configuring automatic client reroute (ACR) ACR seamlessly redirects client applications from a failed server to an alternate server so that these applications can continue their work with minimal interruption.
- Configuring fault monitor Db2 fault monitor keeps Db2 instances up and running by monitoring them and restarting them in the event of unexpected failures.
- Configuring Db2 high availability disaster recovery (HADR) HADR protects you against data loss and downtime from site failures by replicating data changes from a primary database to a standby database.
- Scheduling maintenance activities Through careful planning, automating, and scheduling your maintenance operations, you can help maximize your database's availability.
- Configuring cluster managing software Cluster managing software can help automate the transfer of database operations from a failed primary database to a secondary or standby database.
- Configuring database logging options Use database logging configuration parameters to specify data logging options for your database, such as the type of logging to use, the size of the log files, and the location where log files should be stored.

## Configuring TCP/IP keepalive parameters

Db2 connections between clients and servers use the TCP/IP protocol to communicate. In order to prevent potential failover issues caused by timeouts within the TCP/IP layer, it is necessary to adjust the TCP/IP keepalive parameters on the client. Decreasing the keepalive values on the client improves timely detection of server failures.

### Configuring operating system TCP/IP keepalive parameters for high availability clients

#### About this task

For client systems that do not support the **keepAliveTimeout** parameter because of an operating system, platform or Java Development Kit (JDK) limitation, the TCP/IP keepalive settings can be set at the operating system level by adjusting certain parameters. These clients include Solaris 10.1 Fix Pack 2 and earlier.

For client systems that do support the **keepAliveTimeout** parameter adjusting operating system parameters can also allow for early detection of a non-responsive socket and for faster client reroute.

The values provided in these commands are suggested values, but you should fine-tune these settings based on your specific network and server capabilities.

**Note:** By altering these settings at an operating system level, this will affect all TCP/IP communications on the client.

## Procedure

- **Configuring operating system TCP/IP parameters for clients that support the **keepAliveTimeout** parameter**

If the client is waiting on a receive response from the server, the **keepAliveTimeout** parameter is enabled when the maximum number of retransmissions have finished. To specify the maximum number of retransmissions you can modify the following operating system parameters:

- **On Linux:** The **tcp\_retries2** parameter
- **On Solaris and HP-UX:** The **tcp\_ip\_abort\_interval** parameter
- **On Windows:** The **TcpMaxDataRetransmissions** parameter

- **Configuring operating system TCP/IP parameters for clients that do not support the **keepAliveTimeout** parameter**

*Updating Solaris:* For Solaris 10.1 Fix Pack 2 and earlier clients, change the following operating system keepalive parameter:

- **tcp\_keepalive\_interval** - sets a probe interval that is first sent out after a TCP connection is idle on a system-wide basis.

To display the value of the **tcp\_keepalive\_interval** parameter run the following command:

```
ndd -get /dev/tcp tcp_keepalive_interval
```

To update the value of the **tcp\_keepalive\_interval** parameter run the following command:

```
ndd -set /dev/tcp tcp_keepalive_interval
```

## What to do next

For other client platforms, refer to your operating system documentation on how to set TCP/IP keepalive values.

## Configuring TCP/IP keepalive parameters for high availability clients

The recommended method of setting the keepalive parameters on the client is to use the **keepAliveTimeout** parameter in the **db2dsdriver.cfg** configuration file.

## About this task

The values provided in these commands are suggested values, but you should fine-tune these settings based on your specific network and server capabilities.

## Procedure

There are two methods to update the TCP/IP keepalive parameters:

- Modify the **db2dsdriver.cfg** file.



To set this parameter, edit the `db2dsdriver.cfg` file and place the **keepAliveTimeout** line outside of the `<acr>` section, but still within the `<databases>` parent section. For example:

```
<configuration>
  <dsncollection>
    <dsn alias="D3D" name="D3D" host="DB2PS-member0" port="5912" />
  </dsncollection>
  <databases>
    <database name="D3D" host="DB2PS-member0" port="5912">
      <parameter name="keepAliveTimeout" value="20"/>
      <acr>
        <parameter name="enableAcr" value="true"/>
        <parameter name="enableSeamlessAcr" value="true"/>
        <parameter name="affinityFailbackInterval" value="15"/>
      </acr>
    </database>
  </databases>
  ...
</configuration>
```

This method is recommended because it can be used for both instance-based clients and drivers without an instance. In addition, by utilizing the `db2dsdriver.cfg` file, each individual database can have a different **keepAliveTimeout** setting.

- **Modify the DB2TCP\_CLIENT\_KEEPAKIVE\_TIMEOUT parameter**

The second method for updating the keepalive parameters is to set the **DB2TCP\_CLIENT\_KEEPAKIVE\_TIMEOUT** parameter to detect failures in the TCP/IP communication layer.

To update this parameter, from a command window or terminal on the client, issue this command:

```
db2set DB2TCP_CLIENT_KEEPAKIVE_TIMEOUT=20
```

This value is specified in seconds.

**Note:** While TCP/IP timeout keepalive is also supported for instance attachments, it can only be set using this second method of specifying a value for the **DB2TCP\_CLIENT\_KEEPAKIVE\_TIMEOUT** parameter. Note that automatic client reroute (acr) does not apply in the case of instance attachments.

Setting the operating system level parameters can help with early detection of a non-responsive socket at the remote end. For faster client reroute, configure the TCP/IP keepalive parameter settings at the operating system level.

## Initializing a standby database

One strategy for making a database solution highly available is maintaining a primary database to respond to user application requests, and a secondary or standby database that can take over database operations for the primary database if the primary database fails.

Initializing the standby database entails copying the primary database to the standby database.

### Procedure

There are several ways to initialize the standby database. For example:

- Use disk mirroring to copy the primary database, and use Db2 database suspended I/O support to split the mirror to create the second database.

- Create a backup image of the primary database and recovery that image to the standby database.
- Use SQL replication to capture data from the primary database and apply that data to the standby database.

## What to do next

After initializing the standby database, you must configure your database solution to synchronize the primary database and standby database so the standby database can take over for the primary database if the primary database fails.

## Initializing high availability disaster recovery (HADR)

Use this procedure to set up and initialize Db2 high availability disaster recovery (HADR). Whether you are using a single standby, multiple standbys, or the Db2 pureScale feature, the procedure is similar.

### Before you begin

If you are setting up HADR in a Db2 pureScale environment or if you want to use multiple standby databases, you need to set the **hadr\_target\_list** configuration parameter on all participating databases. This parameter lists the standbys in the scenario when the database becomes a primary. It is required even on a standby. Mutual inclusion is required (that is, if A has B in its target list, B must have A in its target list). This ensures that after a takeover from any standby, the new primary can always keep the old primary as its standby.

If you are configuring multiple standbys, the first standby that you specify in the target list is designated as the *principal HADR standby database*. Additional standbys are *auxiliary HADR standby databases*. The target list need not always include all participants. As well, there is no requirement for symmetry or reciprocity if there is more than one standby; even if you designate that database A has database B as its principal standby, database B does not have to designate A as its principal standby. Each standby specified in the target list of database A, must also have database A in its target list. Working out the target list for each database is an important step.

If you are configuring HADR in a Db2 pureScale environment, you specify a remote cluster with **hadr\_target\_list**. You do not need to list every member in that remote cluster, but you should always include the preferred replay member. For smaller clusters, it is recommended that you include all members, whereas in larger clusters, it is sufficient to include a subset of members as long as the members that are listed are the ones that are most likely to be online.

If you are recovering from a tablespace error or tablespace unavailability on the standby database, refer to *How to recover from tablespace errors on an HADR standby database*.

### About this task

HADR is only supported on a database that is configured with “Archive logging” on page 24. If your database is currently configured with “Circular logging” on page 23, you must first change the `logarchmeth1` and/or the `logarchmeth2` database configuration parameters. An offline backup of the database is required before the database is changed to use archive logging.

HADR can be initialized through the command line processor (CLP), or by calling the db2HADRStart API. The general procedure is to take a backup of the primary, restore it to the standby, set various HADR configuration parameters, and then issue the **START HADR** command. The backup of the primary can be an online backup. As of Db2 Version 11.1.2.2, the backup of the primary can alternatively be a series of table space backup images that are restored using the “Database rebuild” on page 389 feature.

**Note:** This is a generic HADR setup; for more advanced configuration options and settings, see the related links.

## Procedure

To use the CLP to initialize HADR on your system for the first time:

1. Determine the host name, host IP address, and the service name or port number for each of the HADR databases.

If a host has multiple network interfaces, ensure that the HADR host name or IP address maps to the intended one. You need to allocate separate HADR ports in /etc/services for each protected database. These cannot be the same as the ports allocated to the instance. The host name can only map to one IP address.

To determine the host name, see the LIST DATABASE DIRECTORY command. To determine the host IP address, the service name, and port number, see the LIST NODE DIRECTORY command.

**Note:** The instance names for the primary and standby databases do not have to be the same.

2. Set any configuration parameters recommended or required for HADR environments on the primary so that those settings will exist on any standby you create in the next step. For example, enable the recommended logging and index re-creation behavior by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING
    LOGINDEXBUILD    ON
    LOGARCHMETH1     method"
```

3. Create the standby database by restoring a backup image or by initializing a split mirror, based on the existing database that is to be the primary.

Option	Description
Backup and Restore Method	<p>In the following example, the <b>BACKUP DATABASE</b> and <b>RESTORE DATABASE</b> commands are used to initialize a standby database. In this case, a shared file system is accessible at both sites.</p> <ol style="list-style-type: none"> <li>1. On the primary, issue the following command while online:  <code>BACKUP DB <i>dbname</i></code></li> <li>2. If the database already exists on a standby instance, drop it first for a clean start. Files from the existing database can interfere with HADR operation. For example, left over log files can lead the standby onto a log chain not compatible with the primary. Issue the following command to drop the database:  <code>DROP DB <i>dbname</i></code></li> <li>3. On each standby instance, issue the following command :  <code>RESTORE DB <i>dbname</i></code></li> </ol> <p>The following <b>RESTORE DATABASE</b> command options should be avoided when setting up the standby database: <b>TABLESPACE</b>, <b>INTO</b>, <b>REDIRECT</b>, and <b>WITHOUT ROLLING FORWARD</b>.</p> <p>Note: if the primary database is defined over multiple storage paths, the RESTORE command can use the <b>ON path-list</b> option to specify these storage paths. It is important that these paths are listed in the same order as the primary database (the order can be found via <code>db2pd -db <i>dbname</i> -storagepaths</code> command).</p>
Split Mirror Method	<p>The following example illustrates how to use the <b>db2inidb</b> utility to initialize the standby database using a split mirror of the primary database. This procedure is an alternative to the backup and restore procedure illustrated previously.</p> <p>Issue the following command at the standby database:  <code>DB2INIDB <i>dbname</i> AS STANDBY</code></p> <p>Do not use the <b>SNAPSHOT</b> or <b>MIRROR</b> options of db2inidb utility. You can specify the <b>RELOCATE USING</b> option to change one or more of the following configuration attributes: instance name, log path, and database path. However, you must not change the database name or the table space container paths.</p>

**Note:** The database names for the primary and standby databases must be the same.

4. Set the HADR-specific configuration parameters. For Db2 pureScale environments, follow these steps.

- Environments other than Db2 pureScale:

- a. On the primary and standby databases, set the **hadr\_local\_host**, **hadr\_local\_svc**, and **hadr\_syncmode** configuration parameters:

```
"UPDATE DB CFG FOR dbname USING
HADR_LOCAL_HOST    hostname
HADR_LOCAL_SVC     servicename
HADR_SYNCMODE      syncmode"
```

**Note:** When **hadr\_target\_list** is set, the **hadr\_syncmode** is the mode that the principal standby uses when this database becomes a primary. Auxiliary standbys always use SUPERANSYNC for their effective synchronization mode.

- b. On the primary and standby databases, set the **hadr\_target\_list** configuration parameter:

```
UPDATE DB CFG FOR dbname USING
HADR_TARGET_LIST  principalhostname:principalservicename|
                  auxhostnameN:auxservicenameN3
```

If you do not set the **hadr\_target\_list** parameter, you are limited to one standby. This method of configuring HADR is deprecated starting in Version 10.5.

If you are setting up multiple standby databases, the first database that you list is designated as the principal standby.

- c. On the primary and standby databases, set the **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters.

On the primary, set the parameters to the corresponding values on the standby (principal standby if you configure multiple standbys) by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING
HADR_REMOTE_HOST    principalhostname
HADR_REMOTE_SVC     principalservicename
HADR_REMOTE_INST    principalinstname"
```

On the standby, set the parameters to the corresponding values on the primary by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING
HADR_REMOTE_HOST    primaryhostname
HADR_REMOTE_SVC     primaryservicename
HADR_REMOTE_INST    primaryinstname"
```

If you have configured **hadr\_target\_list**, these values are automatically corrected if necessary; however, explicitly setting them to the correct values makes correct values available immediately. These values are used by the IBM Tivoli System Automation for Multiplatforms (SA MP) software to construct the resource names. Thus if you are using SA MP, you must correctly set them before enabling SA MP.

- Db2 pureScale environments:

- a. On the primary and standby databases, set these cluster-level configuration parameters: **hadr\_target\_list** and **hadr\_syncmode**:

```
"UPDATE DB CFG FOR dbname USING
HADR_TARGET_LIST    {memhostname1:memservicename1|
                    memhostnameN:memservicenameN}
HADR_SYNCMODE       syncmode"
```

---

3. You would only specify more than one database in the target list if you are setting up multiple standbys.

The following example shows the command:

```
db2 "UPDATE DB CFG FOR hadr_db USING
    HADR_TARGET_LIST {s0:4000|s1:4000|s2:4000|s3:4000}
    HADR_SYNCMODE      async"
```

The ***hadr\_target\_list*** parameter lists members of the remote cluster. The members of a cluster must be enclosed in braces {}. Only a subset of remote cluster's member addresses are required.

The ***hadr\_remote\_host***, ***hadr\_remote\_svc***, and ***hadr\_remote\_inst*** configuration parameters are automatically configured in Db2 pureScale environments, so they can be left as blank (logically NULL). For more information on automatic configuration, see this section.

- b. On each of the members on the primary and standby databases, set these member-level configuration parameters: ***hadr\_local\_host*** and ***hadr\_local\_svc***:

```
"UPDATE DB CFG FOR dbname MEMBER mname USING
    HADR_LOCAL_HOST memhostname
    HADR_LOCAL_SVC  memservicename"
```

The following examples shows the command:

- For member 0:

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 0 USING
    HADR_LOCAL_HOST p0
    HADR_LOCAL_SVC  4000"
```

- For member 1:

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 1 USING
    HADR_LOCAL_HOST p1
    HADR_LOCAL_SVC  4000"
```

- For member 2:

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 2 USING
    HADR_LOCAL_HOST p2
    HADR_LOCAL_SVC  4000"
```

- For member 3:

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 3 USING
    HADR_LOCAL_HOST p3
    HADR_LOCAL_SVC  4000"
```

5. Connect to the standby instance and start HADR on the standby database. In a Db2 pureScale environment, make sure that you are starting HADR from the member that you want to designate as the preferred replay member.

```
START HADR ON DB dbname AS STANDBY
```

**Note:** Usually, the standby database is started first. If you start the primary database first, this startup procedure will fail if the standby database is not started within the time period specified by the ***hadr\_timeout*** database configuration parameter.

After the standby starts, it enters *local catchup* state in which locally available log files are read and replayed. After it has replayed all local logs, it enters *remote catchup pending* state.

6. Connect to the primary instance and start HADR on the primary database. In a Db2 pureScale environment, make sure you are starting HADR from the member that you want to designate as the preferred replay member.

```
START HADR ON DB dbname AS PRIMARY
```

After the primary starts, the standby enters *remote catchup* state in which receives log pages from the primary and replays them. After it has replayed all

log files that are on the disk of the primary database machine, both databases enter *peer* state (unless SUPERASYNC is the synchronization mode).

## What to do next

Ensure that HADR is up and running by using the `MON_GET_HADR` table function (on the primary or read-enabled standby) or the `db2pd` command with the `-hadr` option.

For more information and examples, see the user scenario Deploying HADR in a Db2 pureScale environment.

## Configuring automatic client reroute and high availability disaster recovery (HADR)

You can use automatic client reroute (ACR) with high availability disaster recovery (HADR) to transfer client application requests from a failed database server to a standby database server.

### Restrictions

- Rerouting is only possible when an alternate database location has been specified at the server.
- Automatic client reroute is only supported with TCP/IP protocol.
- You cannot use ACR if you have client affinity enabled.

### Configuration details

- Use the **UPDATE ALTERNATE SERVER FOR DATABASE** command to enable automatic client reroute.
- Automatic client reroute does not use the `hadr_remote_host` and `hadr_remote_svc` database configuration parameters.
- You can only specify one standby database for automatic client reroute.
- The alternate host location is stored in the system database directory file at the server.
- If automatic client reroute is not enabled, client applications receive error message SQL30081N, and no further attempts are made to establish a connection with the server.

## Setting up automatic client reroute with HADR

The system is set up as follows:

- There is a client where database MUSIC is cataloged as being located at host HORNET with port number 456, which is assigned by the **svcename** configuration parameter.
- Database MUSIC is the primary database and its corresponding standby database, also MUSIC, resides on host MONTERO with port number 456, which is assigned by the **svcename** configuration parameter.

To enable automatic client reroute, update the alternate server for database MUSIC on host HORNET:

```
DB2 UPDATE ALTERNATE SERVER FOR DATABASE music USING HOSTNAME montero PORT 456
```

After this command is issued, the client must successfully connect to host HORNET to obtain the alternate server information. Then, if a communication error occurs between the client and database MUSIC at host HORNET, the client

first attempts to reconnect to database MUSIC at host HORNET. If this fails, the client then attempts to establish a connection with the standby database MUSIC on host MONTERO.

To ensure that ACR can still be used in the event of a role switch, configure the primary database as the alternate server for the standby by issuing the following command on the standby:

```
DB2 UPDATE ALTERNATE SERVER FOR DATABASE music USING HOSTNAME hornet PORT 456
```

### Setting up automatic client reroute with HADR in a Db2 pureScale environment

A user wants to set up ACR for a three-member Db2 pureScale instance that is configured with HADR. The system is set up as follows:

- Database name: `hadr_db`
- Instance owner on all hosts: `db2inst`
- TCP port that is used for HADR primary-standby communication: 4000
- TCP port that is used for SQL client/server communication: 8000
- Hosts for cluster caching facilities (with IDs 128 and 129) and members (with IDs 0, 1, 2, and 3) on the primary: `cfp0`, `cfp1`, `p0`, `p1`, `p2`, and `p3`
- Hosts for cluster caching facilities and members on the standby: `cfs0`, `cfs1`, `s0`, `s1`, `s2`, and `s3`

The user issues the following command from member 0 on the primary:

```
DB2 UPDATE ALTERNATE SERVER FOR DATABASE hadr_db USING HOSTNAME s0 PORT 8000
```

The first time a client connects to the primary, the server returns the addresses of all members in the primary cluster and the alternate server address (`s0:8000`), which is member 0 of the standby cluster. If a client cannot connect to one member on the primary cluster, it tries to connect to another member on the primary. If the client cannot connect to any member on the primary, it tries to connect to member 0 on the standby. The user could further improve availability by using a connection distributor or multi-home DNS entry, which includes multiple members on the standby, as the alternate server address.

To ensure that ACR can still be used in the event of a role switch, the user also configures the primary cluster as the alternate server for the standby by issuing the following command from member 0 on the standby:

```
DB2 UPDATE ALTERNATE SERVER FOR DATABASE hadr_db USING HOSTNAME p0 PORT 8000
```

### Index logging and high availability disaster recovery (HADR)

You should consider setting the database configuration parameters **`logindexbuild`** and **`indexrec`** for high availability disaster recovery (HADR) databases.

#### Using the `logindexbuild` database configuration parameter

**Recommendation:** For HADR databases, set the **`logindexbuild`** database configuration parameter to ON to ensure that complete information is logged for index creation, re-creation, and reorganization. Although this means that index builds might take longer on the primary system and that more log space is required, the indexes will be rebuilt on the standby system during HADR log replay and will be available when a failover takes place. Otherwise, when replaying an index build or rebuild event, the standby marks the index invalid, because the log records do not contain enough information to populate the new



index. If index builds on the primary system are not logged and a failover occurs, any invalid indexes that remain after the failover is complete have to be rebuilt before they can be accessed. While the indexes are being re-created, they cannot be accessed by any applications.

**Note:** If the LOG INDEX BUILD table attribute is set to its default value of NULL, Db2 uses the value specified for the **logindexbuild** database configuration parameter. If the LOG INDEX BUILD table attribute is set to ON or OFF, the value specified for the **logindexbuild** database configuration parameter is ignored.

You might choose to set the LOG INDEX BUILD table attribute to OFF on one or more tables for either of the following reasons:

- You do not have enough active log space to support logging of the index builds.
- The index data is very large and the table is not accessed often; therefore, it is acceptable for the indexes to be re-created at the end of the takeover operation. In this case, set the **indexrec** configuration parameter to RESTART. Because the table is not frequently accessed, this setting causes the system to re-create the indexes at the end of the takeover operation instead of waiting for the first time the table is accessed after the takeover operation.

If the LOG INDEX BUILD table attribute is set to OFF on one or more tables, any index build operation on those tables might cause the indexes to be re-created any time a takeover operation occurs. Similarly, if the LOG INDEX BUILD table attribute is set to its default value of NULL, and the **logindexbuild** database configuration parameter is set to OFF, any index build operation on a table might cause the indexes on that table to be re-created any time a takeover operation occurs. You can prevent the indexes from being re-created by taking one of the following actions:

- After all invalid indexes are re-created on the new primary database, take a backup of the database and apply it to the standby database. As a result of doing this, the standby database does not have to apply the logs used for re-creating invalid indexes on the primary database, which would mark those indexes as rebuild required on the standby database.
- Set the LOG INDEX BUILD table attribute to ON, or set the LOG INDEX BUILD table attribute to NULL and the **logindexbuild** configuration parameter to ON on the standby database to ensure that the index re-creation will be logged.

### Using the **indexrec** database configuration parameter

**Recommendation:** Set the **indexrec** database configuration parameter to RESTART (the default) on both the primary and standby databases. This causes invalid indexes to be rebuilt after a takeover operation is complete. If any index builds have not been logged, this setting allows Db2 to check for invalid indexes and to rebuild them. This process takes place in the background, and the database is accessible after the takeover operation has completed successfully.

If a transaction accesses a table that has invalid indexes before the indexes have been rebuilt by the background re-create index process, the invalid indexes are rebuilt by the first transaction that accesses it.

### Database configuration for high availability disaster recovery (HADR)

You can use database configuration parameters to help achieve optimal performance with Db2 HADR.

In most cases, use the same database configuration parameter settings and database manager configuration parameter settings on the systems where the primary and standby databases are located. If the settings for the configuration parameters on the standby database are different from the settings on the primary, the following problems might occur:

- Error messages might be returned for the standby database while the log files that were shipped from the primary database are being replayed.
- After a takeover operation, the new primary database might be unable to handle the workload, resulting in performance problems or in applications receiving error messages that they did not receive when they were connected to the original primary database.

Changes to the configuration parameters on the primary database are not automatically propagated to the standby database. You must manually make changes on the standby database. For dynamic configuration parameters, changes take effect without having to shut down and restart the database management system (DBMS) or the database. For non-dynamic configuration parameters, changes take effect after the standby database is restarted.

Following are sections on specific configuration topics for HADR:

- “Size of log files configuration parameter on the standby database”
- “Database configuration parameter `autorestart`” on page 119
- “Log receive buffer size on a standby database” on page 119
- “Load operations and HADR” on page 119
- “`DB2_HADR_PEER_WAIT_LIMIT` registry variable” on page 120
- “HADR configuration parameters” on page 121

### Size of log files configuration parameter on the standby database

One exception to the configuration parameter behavior that is described in the previous paragraph is the behavior of the **`logfilsiz`** database configuration parameter. Although the value of this parameter is not replicated to the standby database, to help ensure that there are identical log files on both databases, the setting for the **`logfilsiz`** configuration parameter on the standby is ignored. Instead, the database creates local log files whose sizes match the size of the log files on the primary database.

After a takeover, the original standby (new primary) uses the **`logfilsiz`** parameter value that you set on the original primary until you restart the database. At that point, the new primary reverts to using the value that you set locally. In addition, the current log file is truncated and any pre-created log files are resized on the new primary.

If the databases keep switching roles as a result of a non-forced takeover and neither database is deactivated, the log file size that is used is always the one from the original primary database. However, if there is a deactivation and then a restart on the original standby (new primary), the new primary uses the log file size that you configured locally. This log file size continues to be used if the original primary takes over again. Only after a deactivation and restart on the original primary would the log file size revert to the settings on the original primary.

## Database configuration parameter **autorestart**

The recommended configuration for the **autorestart** parameter on HADR systems is ON. If the **autorestart** parameter is set to OFF, and the server fails, your response depends on whether you want to restart or fail over to the standby:

- If you want to restart, run the **RESTART DATABASE** command manually. If the restart fails, perform failover.
- If you want to fail over, perform the following steps:
  1. Shut down the old primary to prevent a “split brain”. Do this by either stopping the Db2 instance or powering off the host machine. If the server is not accessible for administration, fence it off from clients by disabling the client/server network.

**Note:** Deactivating the database is not sufficient because client connections can bring it back online. If it failed in a consistent state, then even if the **autorestart** parameter is set to OFF, client connections can bring it back online.

2. After you shut down old primary, issue the **TAKEOVER HADR** command with the BY FORCE option on the standby.

## Log receive buffer size on a standby database

By default, the log receive buffer size on a standby database is two times the value that you specify for the **logbufsz** configuration parameter on the primary database. This size might not be sufficient. For example, consider what might happen when the HADR synchronization mode is set to ASYNC and the primary and standby databases are in peer state. If the primary database is also experiencing a high transaction load, the log receive buffer on the standby database might fill to capacity, and the log shipping operation from the primary database might stall. To manage these temporary peaks, you can make either of the following configuration changes:

- Increase the size of the log receive buffer on the standby database by modifying the value of the **DB2\_HADR\_BUF\_SIZE** registry variable.
- Enable log spooling on a standby database by setting the **hadr\_spool\_limit** configuration parameter.

## Load operations and HADR

If you issue the **LOAD** command on the primary database with the **COPY YES** parameter, the command executes on the primary database, and the data is replicated to the standby database if the load copy can be accessed through the path or device that is specified by the command. If load copy data cannot be accessed from the standby database, the table space in which the table is stored is marked invalid on the standby database. Any future log records that pertain to this table space are skipped. To ensure that the load operation can access the load copy on the standby database, use a shared location for the output file from the **COPY YES** parameter. Alternatively, you can deactivate the standby database while performing the load on the primary, place a copy of the output file in the standby path, and then activate the standby database.

If you issue the **LOAD** command with the **NONRECOVERABLE** parameter on the primary database, the command executes on the primary database, and the table on the standby database is marked invalid. Any future log records that pertain to this

table are skipped. You can issue the **LOAD** command with the **COPY YES** and **REPLACE** parameters to bring the table back, or you can drop the table to recover the space.

**Note:** You cannot bring a table back using the **LOAD** command with the **COPY YES** and **REPLACE** options if the table has one of the following characteristics:

- The table was created with the **NOT LOGGED INITIALLY** attribute.
- The table is a multidimensional clustered (MDC) table.
- The table has compression dictionaries.
- The table has XML columns.

Because a load operation with the **COPY NO** parameter is not supported with HADR, the operation is automatically converted to a load operation with the **NONRECOVERABLE** parameter. To enable a load operation with the **COPY NO** parameter to be converted to a load operation with the **COPY YES** parameter, set the **DB2\_LOAD\_COPY\_NO\_OVERRIDE** registry variable on the primary database. This registry variable is ignored on the standby database. Ensure that the device or directory that you specify for the primary database can be accessed by the standby database by using the same path, device, or load library.

If you are using the Tivoli Storage Manager (TSM) software to perform a load operation with the **COPY YES** parameter, you might have to set the **vendoropt** configuration parameter on the primary and standby databases. Depending on how you configured TSM, the values on the primary and standby databases might not be the same. Also, when using TSM to perform a load operation with the **COPY YES** parameter, you must issue the **db2adutl** command with the **GRANT** parameter to give the standby database read access to the files that are loaded.

If table data is replicated by a load operation with the **COPY YES** parameter, the indexes are replicated as follows:

- If you specify the **REBUILD** indexing mode option with the **LOAD** command and the **LOG INDEX BUILD** table attribute is set to **ON** (using the **ALTER TABLE** statement), or if it is set to **NULL** and the **logindexbuild** database configuration parameter is set to **ON**, the primary database includes the rebuilt index object (that is, all of the indexes defined on the table) in the copy file to enable the standby database to replicate the index object. If the index object on the standby database is marked invalid before the load operation, it becomes usable again after the load operation as a result of the index rebuild.
- If you specify the **INCREMENTAL** indexing mode option with the **LOAD** command and the **LOG INDEX BUILD** table attribute is set to **ON** (using the **ALTER TABLE** statement), or if it is set to **NULL** and the **logindexbuild** database configuration parameter on the primary database is set to **ON**, the index object on the standby database is updated only if it is not marked invalid before the load operation. Otherwise, the index is marked invalid on the standby database.

### **DB2\_HADR\_PEER\_WAIT\_LIMIT registry variable**

**Restriction:** None of this section applies to auxiliary standbys because they are in **SUPERASYNC** synchronization mode, so they do not ever enter peer state.

If you set the **DB2\_HADR\_PEER\_WAIT\_LIMIT** registry variable, the HADR primary database breaks out of peer state if logging on the primary database has been blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database breaks the connection to the standby database. If you disable the peer window by setting the

**hadr\_peer\_window** configuration parameter to 0, the primary enters the disconnected state, and logging resumes. If you enable the peer window, the primary database enters disconnected peer state, in which logging continues to be blocked. The primary leaves disconnected peer state upon reconnection or peer window expiration. Logging resumes after the primary leaves disconnected peer state.

**Note:** If you set **DB2\_HADR\_PEER\_WAIT\_LIMIT**, use a minimum value of 10 to avoid triggering false alarms.

Honoring peer window transition when a database breaks out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies: safe takeover from the standby if it is still in disconnected peer state.

On the standby side, after disconnection, the database continues replaying already received logs. After the received logs have been replayed, the standby reconnects to the primary. After replaying the received logs, the standby reconnects to the primary. Upon reconnection, normal state transition follows: first remote catchup state, then peer state.

#### Relationship to **hadr\_timeout** database configuration parameter

The **hadr\_timeout** database configuration parameter does not break the primary out of peer state if the primary keeps receiving heartbeat messages from the standby while blocked. The **hadr\_timeout** database configuration parameter specifies a timeout value for the HADR network layer. An HADR database breaks the connection to its partner database if it has not received any message from its partner for the period that is specified by the **hadr\_timeout** configuration parameter. The timeout does not control timeout for higher-layer operations such as log shipping and ack (acknowledgement) signals. If log replay on the standby database is stuck on a large operation such as load or reorganization, the HADR component still sends heartbeat messages to the primary database on the normal schedule. In such a scenario, the primary is blocked as long as the standby replay is blocked unless you set the **DB2\_HADR\_PEER\_WAIT\_LIMIT** registry variable.

The **DB2\_HADR\_PEER\_WAIT\_LIMIT** registry variable unblocks primary logging regardless of connection status. Even if you do not set the **DB2\_HADR\_PEER\_WAIT\_LIMIT** registry variable, the primary always breaks out of peer state when a network error is detected or the connection is closed, possibly as result of the **hadr\_timeout** configuration parameter.

#### HADR configuration parameters

Some HADR configuration parameters are static, such as **hadr\_local\_host** and **hadr\_remote\_host**. Static parameters are loaded on database startup, and changes are ignored during run time. HADR parameters are also loaded when the **START HADR** command completes. On the primary database, HADR can be started and stopped dynamically, with the database remaining online. Thus, one way to refresh the effective value of an HADR configuration parameter without shutting down the database is to stop and restart HADR. In contrast, the **STOP HADR** brings down the database on the standby, so the standby's parameters cannot be refreshed with the database online.

## Host name parameters and service and port name parameters

There are six interrelated configuration parameters that you need to set for HADR:

- **hadr\_target\_list**
- **hadr\_local\_host**
- **hadr\_remote\_host**
- **hadr\_local\_svc**
- **hadr\_remote\_svc** (except in a Db2 pureScale environment, which does not make use of this parameter)
- **hadr\_remote\_inst**

The target list specifies a set of host:port pairs that act as standbys (for the primary) or the standby hosts to be used if the standby takes over as the new HADR primary database. For a detailed description of its usage, see the **hadr\_target\_list** topic in the Related links.

TCP connections are used for communication between the primary and standby databases. The “local” parameters specify the local address and the “remote” parameters specify the remote address. A primary database or primary database member listens on its local address for new connections. A standby database that is not connected to a primary database retries connection to its remote address.

The standby database also listens on its local address. In some scenarios, another HADR database can contact the standby database on this address and send it messages.

Unless the **HADR\_NO\_IP\_CHECK** registry variable is set, HADR does the following cross-checks of the primary and principal standbys' local and remote addresses on connection:

*my local address = your remote address*

and

*my remote address = your local address*

The check is done using the IP address and port number, rather than the literal string in the configuration parameters. You need to set the **HADR\_NO\_IP\_CHECK** registry variable in NAT (Network Address Translation) environment to bypass the check.

You can configure an HADR database to use either IPv4 or IPv6 to locate its partner database. If the host server does not support IPv6, you must use IPv4. If the server supports IPv6, whether the database uses IPv4 or IPv6 depends upon the format of the address that you specify for the **hadr\_local\_host** and **hadr\_remote\_host** configuration parameters. The database attempts to resolve the two parameters to the same IP format and use IPv6 when possible. Table 9 shows how the IP mode is determined for IPv6-enabled servers:

Table 9. How the address space used for HADR communication is determined

IP mode used for <b>hadr_local_host</b> parameter	IP mode used for <b>hadr_remote_host</b> parameter	IP mode used for HADR communications
IPv4 address	IPv4 address	IPv4
IPv4 address	IPv6 address	Error

Table 9. How the address space used for HADR communication is determined (continued)

IP mode used for <b>hadr_local_host</b> parameter	IP mode used for <b>hadr_remote_host</b> parameter	IP mode used for HADR communications
IPv4 address	host name, maps to IPv4 only	IPv4
IPv4 address	host name, maps to IPv6 only	Error
IPv4 address	host name, maps to IPv4 and v6	IPv4
IPv6 address	IPv4 address	Error
IPv6 address	IPv6 address	IPv6
IPv6 address	host name, maps to IPv4 only	Error
IPv6 address	host name, maps to IPv6 only	IPv6
IPv6 address	host name, maps to IPv4 and IPv6	IPv6
hostname, maps to IPv4 only	IPv4 address	IPv4
hostname, maps to IPv4 only	IPv6 address	Error
hostname, maps to IPv4 only	hostname, maps to IPv4 only	IPv4
hostname, maps to IPv4 only	hostname, maps to IPv6 only	Error
hostname, maps to IPv4 only	hostname, maps to IPv4 and IPv6	IPv4
hostname, maps to IPv6 only	IPv4 address	Error
hostname, maps to IPv6 only	IPv6 address	IPv6
hostname, maps to IPv6 only	hostname, maps to IPv4 only	Error
hostname, maps to IPv6 only	hostname, maps to IPv6 only	IPv6
hostname, maps to IPv6 only	hostname, maps to IPv4 and IPv6	IPv6
hostname, maps to IPv4 and IPv6	IPv4 address	IPv4
hostname, maps to IPv4 and IPv6	IPv6 address	IPv6
hostname, maps to IPv4 and IPv6	hostname, maps to IPv4 only	IPv4
hostname, maps to IPv4 and IPv6	hostname, maps to IPv6 only	IPv6
hostname, maps to IPv4 and IPv6	hostname, maps to IPv4 and IPv6	IPv6

The primary and standby databases can make HADR connections only if they use the same IPv4 or IPv6 format. If one server is IPv6 enabled (but also supports IPv4) and the other server supports IPv4 only, at least one of the **hadr\_local\_host** and **hadr\_remote\_host** parameters on the IPv6 server must specify an IPv4 address to force database on this server to use IPv4.

You can set the HADR local service and remote service parameters (**hadr\_local\_svc** and **hadr\_remote\_svc**) to either a port number or a service name. The values that you specify must map to ports that are not being

used by any other service, including other Db2 components or other HADR databases. In particular, you cannot set either parameter value to the TCP/IP port that is used by the server to await communications from remote clients (the value of the **svcname** database manager configuration parameter) or the next port (the value of the **svcname** parameter + 1).

If the primary and standby databases are on different servers, they can use the same port number or service name; otherwise, they must have different values.

### Automatic reconfiguration

The **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters are automatically reset when HADR starts if you did not correctly set them. Even though this automatic reconfiguration occurs, always try to set the correct initial values because that reconfiguration might not take effect until a connection is made between a standby and its primary. In some HADR deployments, those initial values might be needed. For example, if you are using the IBM Tivoli System Automation for Multiplatforms software, the value for the **hadr\_remote\_inst** configuration parameter is needed to construct a resource name.

**Note:** If the **DB2\_HADR\_NO\_IP\_CHECK** registry variable is set to ON, the **hadr\_remote\_host** and **hadr\_remote\_svc** are not automatically updated.

Reconfiguration is predicated on the values of the **hadr\_target\_list** configuration parameter being correct; if anything is wrong in a target list entry, you must correct it manually.

On the primary, the reconfiguration occurs in the following manner:

- If the values for the **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters do not match the *host:port* pair that is the first entry of the **hadr\_target\_list** configuration parameter (namely, the principal standby), the **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters are updated with the values from the target list.
- If the value for the **hadr\_remote\_inst** configuration parameter does not match the instance name of the principal standby, the correct instance name is copied to the **hadr\_remote\_inst** configuration parameter for the primary after the principal standby connects to it.

On a standby database, the reconfiguration occurs in the following manner:

- When a standby starts, it attempts to connect to the database that you specified for its **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters.
- If the standby cannot connect to the primary, it waits for the primary to connect to it.
- The primary attempts to connect to its standbys using addresses listed in its **hadr\_target\_list** parameter. After the primary connects to a standby, the **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters for the standby are updated with the correct values for the primary.

In a non-forced takeover, the values for the **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters on the new primary are automatically updated to its principal standby, and these parameters on the standbys listed in the new primary's **hadr\_target\_list**



are automatically updated to point to the new primary. Any database that is not listed in the new primary's **hadr\_target\_list** is not updated. Those databases continue to attempt to connect to the old primary and get rejected because the old primary is now a standby. The old primary is guaranteed to be in the new primary's target list because of the requirement of mutual inclusion in the target list.

In a forced takeover, automatic update on the new primary and its standbys (excluding the old primary) work the same way as non-forced takeover. However, automatic update on the old primary does not happen until it is shut down and restarted as a standby for reintegration.

Any database that is not online during the takeover will be automatically reconfigured after it starts. Automatic reconfiguration might not take effect immediately on startup, because it relies on the new primary to periodically contact the standby. On startup, a standby might attempt to connect to the old primary and follow the log stream of the old primary, causing it to diverge from the new primary's log stream and, making that standby unable to pair with the new primary. As a result, you must shut down the old primary before takeover to avoid that kind of *split brain* scenario.

### Synchronization mode

The setting for the **hadr\_syncmode** configuration parameter does not have to be the same on the primary and standby databases. Whatever setting you specify for the **hadr\_syncmode** configuration parameter on a standby is considered its *configured synchronization mode*; this setting has relevance only if the standby becomes a primary. The standby is assigned an *effective synchronization mode*. For any auxiliary standby, the effective synchronization mode is always SUPERASYNC. For the principal standby, the effective synchronization mode is the setting for the **hadr\_syncmode** configuration parameter for the primary. For a standby, the monitoring interfaces display the effective synchronization mode as the synchronization mode.

**Note:** If you have set up HADR without using the **hadr\_target\_list** configuration parameter (a method that is deprecated starting in V10.5), the **hadr\_syncmode** configuration parameter must be identical on the primary and standby databases.

For more detailed information, see “Db2 high availability disaster recovery (HADR) synchronization mode”.

### HADR timeout and peer window

The timeout period, which you specify with the **hadr\_timeout** configuration parameter, must be identical on the primary and standby databases. The consistency of the values of these configuration parameters is checked when an HADR pair establishes a connection.

With one exception, when the primary database starts, it waits for the longer of the two following periods for a standby to connect:

- For a minimum of 30 seconds
- For the number of seconds that is specified by the **hadr\_timeout** database configuration parameter.

If the principal standby does not connect in the specified time, the startup fails; a connection to an auxiliary standby is optional. The one exception to this behavior is when you issue the **START HADR** command with the **BY**

**FORCE** parameter. In this case, the primary database starts without waiting for the standby database to connect to it.

After an HADR pair establishes a connection, they exchange heartbeat messages. The heartbeat interval is computed from factors like the **hadr\_timeout** and **hadr\_peer\_window** configuration parameters. It is reported by the `HEARTBEAT_INTERVAL` field in the `MON_GET_HADR` table function and the `db2pd` command. If one database does not receive any message from the other database within the number of seconds that is specified by the **hadr\_timeout** configuration parameter, it initiates a disconnect. This behavior means that at most, it takes the number of seconds that is specified by the **hadr\_timeout** configuration parameter for an HADR database to detect the failure of either its partner database or the intervening network. If you set the **hadr\_timeout** configuration parameter too low, you might receive false alarms and frequent disconnections.

The setting for the **hadr\_peer\_window** configuration parameter does not have to be the same on the primary and standby databases; the principal standby uses the peer window setting of the primary. The exception to this is if you set up HADR without using the **hadr\_target\_list** configuration parameter (a method that is deprecated starting in V10.5), in which case, the **hadr\_peer\_window** configuration parameter must be identical on the primary and standby databases.

Peer window cannot be enabled (that is, it must be set to 0) in the following situations:

- If you are using the Db2 pureScale feature
- If the **hadr\_syncmode** parameter is set to `ASYNCR` or `SUPERASYNCR`
- If you are configuring an auxiliary standby

If you have the **hadr\_peer\_window** configuration parameter set to a nonzero value and the primary loses connection to the standby in peer state, the primary database does not commit transactions until the connection with the standby database is restored or the time value of the **hadr\_peer\_window** configuration parameter elapses, whichever happens first.

For maximal availability, the default value for the **hadr\_peer\_window** database configuration parameter is 0. When this parameter is set to 0, as soon as the connection between the primary and the standby is closed, the primary drops out of peer state to avoid blocking transactions. The connection can close because the standby closed the connection, a network error is detected, or timeout is reached. For increased data consistency, but reduced availability, you can set the **hadr\_peer\_window** database configuration parameter to a nonzero value.

For more information, see “Setting the **hadr\_timeout** and **hadr\_peer\_window** database configuration parameters”.

The following sample configuration is for the primary and standby databases:

Primary database:

<code>HADR_TARGET_LIST</code>	<code>host2.ibm.com:hadr_service</code>
<code>HADR_LOCAL_HOST</code>	<code>host1.ibm.com</code>
<code>HADR_LOCAL_SVC</code>	<code>hadr_service</code>
<code>HADR_REMOTE_HOST</code>	<code>host2.ibm.com</code>
<code>HADR_REMOTE_SVC</code>	<code>hadr_service</code>

HADR_REMOTE_INST	dbinst2
HADR_TIMEOUT	120
HADR_SYNCMODE	NEARSYNC
HADR_PEER_WINDOW	120

Standby database:

HADR_TARGET_LIST	host1.ibm.com:hadr_service
HADR_LOCAL_HOST	host2.ibm.com
HADR_LOCAL_SVC	hadr_service
HADR_REMOTE_HOST	host1.ibm.com
HADR_REMOTE_SVC	hadr_service
HADR_REMOTE_INST	dbinst1
HADR_TIMEOUT	120
HADR_SYNCMODE	NEARSYNC
HADR_PEER_WINDOW	120

### Setting the **hadr\_timeout** and **hadr\_peer\_window** database configuration parameters:

You can configure the **hadr\_timeout** and **hadr\_peer\_window** database configuration parameters for optimal response to a connection failure.

#### **hadr\_timeout** database configuration parameter

If an HADR database does not receive any communication from its partner database for longer than the length of time that is specified by the **hadr\_timeout** database configuration parameter, then the database concludes that the connection with the partner database is lost. If the database is in peer state when the connection is lost, then it moves into disconnected peer state if the **hadr\_peer\_window** database configuration parameter is greater than zero, or into remote catchup pending state if **hadr\_peer\_window** is not greater than zero. The state change applies to both primary and standby databases.

#### **hadr\_peer\_window** database configuration parameter

The **hadr\_peer\_window** configuration parameter does not replace the **hadr\_timeout** configuration parameter. The **hadr\_timeout** configuration parameter determines how long an HADR database waits before it considers that its connection with the partner database as failed. The **hadr\_peer\_window** configuration parameter determines whether the database goes into disconnected peer state after the connection is lost, and how long the database remains in that state. HADR breaks the connection as soon as a network error is detected during send, receive, or poll on the TCP socket. HADR polls the socket every 100 milliseconds. This frequency allows it to respond quickly to network errors detected by the OS. Only in the worst case does HADR wait until the timeout to break a bad connection. In this case, a database application that is running at the time of failure can be blocked for the time equal to the sum of the **hadr\_timeout** and **hadr\_peer\_window** database configuration parameters.

**Note:** The HADR peer window is not supported in a Db2 pureScale environment. Attempts to update it to a nonzero value fail with a warning, and the **START HADR** command fails if **hadr\_peer\_window** is not set to 0.

### Setting the **hadr\_timeout** and **hadr\_peer\_window** database configuration parameters

It is desirable to keep the waiting time that a database application experiences to a minimum. Setting the **hadr\_timeout** and **hadr\_peer\_window** configuration parameters to small values would reduce the time that a database application must wait if an HADR standby database loses its

connection with the primary database. However, you should also consider the following details when you are choosing values to assign to the **hadr\_timeout** and **hadr\_peer\_window** configuration parameters:

- Set the **hadr\_timeout** database configuration parameter to a value that is long enough to avoid false alarms on the HADR connection that are caused by short, temporary network interruptions. For example, the default value of **hadr\_timeout** is 120 seconds, which is a reasonable value on many networks.
- Set the **hadr\_peer\_window** database configuration parameter to a value that is long enough to allow the system to perform automated failure responses. If the HA system, for example a cluster manager, detects primary database failure before disconnected peer state ends, a failover to the standby database takes place. Data is not lost in the failover as all data from old primary is replicated to the new primary. If the peer window is too short, the HA system might not have enough time to detect the failure and respond.

**Note:** The principal standby uses the primary's setting for **hadr\_peer\_window** (the *effective peer window*). The setting for **hadr\_peer\_window** on any auxiliary standby is meaningless because that type of standby always runs in SUPERASYNC mode.

### HADR log spooling:

The high availability disaster recovery (HADR) log spooling feature allows transactions on primary to make progress without having to wait for the log replay on the standby.

When this feature is enabled, log data sent by the primary is *spooled*, or written, to disk on the standby, and that log data is later read by log replay.

Log spooling, which is enabled by setting the **hadr\_spool\_limit** database configuration parameter, is an improvement to the HADR feature. When replay is slow, it is possible that new transactions on the primary can be blocked because it is not able to send log data to the standby system if there is no room in the buffer to receive the data. The log spooling feature means that the standby is not limited by the size of its buffer. When there is an increase in data received that cannot be contained in the buffer, the log replay reads the data from disk. This allows the system to better tolerate either a spike in transaction volume on the primary, or a slow down of log replay (due to the replay of particular type of log records) on the standby.

This feature could potentially lead to a larger gap between the log position of received logs on the standby and the log replay position on the standby, which can lead to longer takeover time. Use the **db2pd** command with the **-hadr** option or the **MON\_GET\_HADR** table function to monitor this gap by comparing the **STANDBY\_LOG\_POS** field, which shows receive position, and the **STANDBY\_REPLAY\_LOG\_POS** field. You should consider your spool limit setting carefully because the old standby cannot start up as the new primary and receive transactions until the replay of the spooled logs has finished.

### Log archiving configuration for Db2 high availability disaster recovery (HADR)

To use log archiving with Db2 high availability disaster recovery (HADR), configure both the primary database and the standby database for automatic log

retrieval capability from all log archive locations. For multiple standby systems, configure archiving on primary and all standby databases.

Only the current primary database can perform log archiving. If the primary and standby databases are set up with separate archiving locations, logs are archived only to the primary database's archiving location. In the event of a takeover, the standby database becomes the new primary database and any logs archived from that point on are saved to the original standby database's archiving location. In such a configuration, logs are archived to one location or the other, but not both; with the exception that following a takeover, the new primary database might archive a few logs that the original primary database had already archived. In a multiple standby system, the archived log files can be scattered among all databases' (primary and standbys) archive devices. A shared archive is preferred because all files are stored in a single location.

Many operations need to retrieve archived log files. These operations include: database roll forward, the HADR primary database retrieving log files to send to the standby database in remote catch up, and replication programs (such as Q Replication) reading logs. As a result, a shared archive for an HADR system is preferred, otherwise, the needed files can be distributed on multiple archive devices, and user intervention is needed to locate the needed files and copy them to the requesting database. The recommended copy destination is an archive device. If copying into an archive is not feasible, copy the logs into the overflow log path. As a last resort, copy them into the log path (but you should be aware that there is a risk of damaging the active log files). Db2 does not auto delete user copied files in the overflow and active log path, so you should manually remove the files when they are no longer needed by any HADR standby or any application.

A specific scenario is a takeover with multiple HADR standbys. After the takeover, the new primary might not have all log files needed by other standbys (because a standby is at an older log position). If the primary cannot find a requested log file, it notifies the standby, which closes the connection and then reconnects in a few seconds to retry. The retry duration is limited to a few minutes. When retry time is exhausted, the standby shuts down. In this case, you should copy the files to the primary to ensure it has files from the first missing file to its current log file. After the files are copied, restart the standby if needed.

The standby database automatically manages log files in its log path. The standby database does not delete a log file from its local log path until it has been notified by the primary database that the primary database has archived it. This behavior provides added protection against the loss of log files. If the primary database fails and its log disk becomes corrupted before a particular log file is archived on the primary database, the standby database does not delete that log file from its own disk because it has not received notification that the primary database successfully archived the log file. If the standby database then takes over as the new primary database, it archives that log file before recycling it. If both the **logarchmeth1** and **logarchmeth2** configuration parameters are in use, the standby database does not recycle a log file until the primary database has archived it using both methods.

In addition to the benefits previously listed, a shared log archive device improves the catchup process by allowing the standby database to directly retrieve older log files from the archive in local catchup state, instead of retrieving those files indirectly through the primary in remote catchup state. However, it is recommended that you not use a serial archive device such as a tape drive for HADR databases. With serial devices, you might experience performance

degradation on both the primary and standby databases because of mixed read and write operations. The primary writes to the device when it archives log files and the standby reads from the device to replay logs. This performance impact can occur even if the device is not configured as shared.

## Shared log archives on Tivoli Storage Manager

Using a shared log archive with IBM Tivoli Storage Manager (TSM) allows one or more nodes to appear as a single node to the TSM server, which is especially useful in an HADR environment where either machine can be the primary at any one time.

To set up a shared log archive, you need to use proxy nodes which allow the TSM client nodes to perform data protection operations against a centralized name space on the TSM server. The target client node owns the data and agent nodes act on behalf of the target nodes to manage the backup data. The proxy node target is the node name defined on the TSM server to which backup versions of distributed data are associated. The data is managed in a single namespace on the TSM server as if it is entirely the data for this node. The proxy node target name can be a real node (for example, one of the application hosts) or a virtual node name (that is, with no corresponding physical node). To create a virtual proxy node name, use the following commands on the TSM server:

```
Grant proxynode target=virtual-node-name agent=HADR-primary-name
Grant proxynode target=virtual-node-name agent=HADR-standby-name
```

Next, you need to set these database configuration parameters on the primary and standby databases to the *virtual-node-name*:

- **vendoropt**
- **logarchopt**

In a multiple standby setup, you need to grade proxynode access to all machines on the TSM server and configure the **vendoropt** and **logarchopt** configuration parameters on all of the standbys.

## High availability disaster recovery (HADR) performance

Configuring different aspects of your database system, including network bandwidth, CPU power, and buffer size, can improve the performance of your Db2 high availability disaster recovery (HADR) databases.

The network is the key part of your HADR setup because network connectivity is required to replicate database changes from the primary to the standby, keeping the two databases in sync.

### Recommendations for maximizing network performance:

- Ensure that network bandwidth is greater than the database log generation rate.
- Consider network delays when you choose the HADR synchronization mode. Network delays affect the primary only in SYNC and NEARSYNC modes.

The slowdown in system performance as a result of using SYNC mode can be significantly larger than that of the other synchronization modes. In SYNC mode, the primary database sends log pages to the standby database only after the log pages are successfully written to the primary database log disk. To protect the integrity of the system, the primary database waits for an acknowledgment from the standby before it notifies an application that a transaction was prepared or committed. The standby database sends the

acknowledgment only after it writes the received log pages to the standby database disk. The performance overhead equals the time that is needed for writing the log pages on the standby database plus the time that is needed for sending the messages back to the primary.

In NEARSYNC mode, the primary database writes and sends log pages in parallel. The primary then waits for an acknowledgment from the standby. The standby database acknowledges as soon as the log pages are received into its memory. On a fast network, the overhead to the primary database is minimal. The acknowledgment might have already arrived by the time the primary database finishes local log write.

For ASYNC mode, the log write and send are also in parallel; however, in this mode the primary database does not wait for an acknowledgment from the standby. Therefore, network delay is not an issue. Performance overhead is even smaller with ASYNC mode than with NEARSYNC mode.

For SUPERASYNC mode, transactions are never blocked or experience elongated response times because of network interruptions or congestion. New transactions can be processed as soon as previously submitted transactions are written to the primary database. Therefore, network delay is not an issue. The elapsed time for the completion of non-forced takeover operations might be longer than in other modes because the log gap between the primary and the standby databases might be relatively larger.

- Consider tuning the **DB2\_HADR\_SOSNDBUF** and **DB2\_HADR\_SORCVBUF** registry variables.

HADR log shipping workload, network bandwidth, and transmission delay are important factors to consider when you are tuning the TCP socket buffer sizes. Two registry variables, **DB2\_HADR\_SOSNDBUF** and **DB2\_HADR\_SORCVBUF** allow tuning of the TCP socket send and receive buffer size for HADR connections only. These two variables have the value range of 1024 to 4294967295 and default to the socket buffer size of the operating system, which varies depending on the operating system. It is strongly recommended that you use a minimum value of 16384 (16 K) for your **DB2\_HADR\_SOSNDBUF** and **DB2\_HADR\_SORCVBUF** settings. Some operating systems automatically round or silently cap the user specified value.

You can use the HADR simulator (a command-line tool that generates a simulated HADR workload) to measure network performance and to experiment with various network tuning options. You can download the simulator at <https://www.ibm.com/developerworks/community/wikis/home/wiki/DB2HADR/page/HADR%20simulator>.

## Network congestion

For each log write on the primary, the same log pages are also sent to the standby. Each write operation is called a *flush*. The size of the flush is limited to the log buffer size on the primary database (which is controlled by the database configuration parameter **logbufsz**). The exact size of each flush is nondeterministic. A larger log buffer does not necessarily lead to a larger flush size.

If the standby database is too slow replaying log pages, its log-receiving buffer might fill up, thereby preventing the buffer from receiving more log pages. In SYNC and NEARSYNC modes, if the primary database flushes its log buffer one more time, the data is likely to be buffered in the network pipeline consisting of the primary machine, the network, and the standby database. Because the standby

database does not have free buffer to receive the data, it cannot acknowledge, so the primary database becomes blocked while it is waiting for the standby database's acknowledgment.

In ASYNC mode, the primary database continues to send log pages until the pipeline fills up and it cannot send additional log pages. This condition is called *congestion*. Congestion is reported by the **hadr\_connect\_status** monitor element. For SYNC and NEARSYNC modes, the pipeline can usually absorb a single flush and congestion does not occur. However, the primary database remains blocked waiting for an acknowledgment from the standby database on the flush operation.

Congestion can also occur if the standby database is replaying log records that take a long time to replay, such as database or table reorganization log records.

In SUPERASYNC mode, since the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap as it is an indirect measure of the potential number of transactions that might be lost if a true disaster occurs on the primary system. In disaster recovery scenarios, any transactions that are committed during the log gap would not be available to the standby database. Therefore, monitor the log gap by checking the **HADR\_LOG\_GAP** field in the output of the **MON\_GET\_HADR** table function or the **db2pd** command with the **-hadr** option; if the size of the log gap is not acceptable, investigate the network interruptions or the relative speed of the standby HADR server and take corrective measures to reduce the log gap.

#### **Recommendations for minimizing network congestion:**

- Use a standby database that is powerful enough to replay the logged operations of the database as fast as they are generated on the primary. Identical primary and standby hardware is recommended.
- Consider tuning the size of the standby database log-receiving buffer by using the **DB2\_HADR\_BUF\_SIZE** registry variable.

A larger buffer can help to reduce congestion, although it might not remove all of the causes of congestion. By default, the size of the standby database log-receiving buffer is two times the size of the primary database log-writing buffer. The database configuration parameter **logbufsz** specifies the size of the primary database log-writing buffer.

You can determine if the standby's log-receiving buffer is inadequate by using the **db2pd** command with the **-hadr** option or the **MON\_GET\_HADR** table function. If the value for the **STANDBY\_RECV\_BUF\_PERCENT** field, which indicates the percentage of standby log receiving buffer that is being used, is close to 100, increase the **DB2\_HADR\_BUF\_SIZE** setting.

- Consider setting the **DB2\_HADR\_PEER\_WAIT\_LIMIT** registry variable, which allows you to prevent primary database logging from blocking because of a slow or blocked standby database.

When the **DB2\_HADR\_PEER\_WAIT\_LIMIT** registry variable is set, the HADR primary database breaks out of the peer state if logging on the primary database is blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database breaks the connection to the standby database. If the peer window is disabled, the primary enters disconnected state and logging resumes. If the peer window is enabled, the primary database enters disconnected peer state, in which logging continues to be blocked. The primary database leaves



disconnected peer state upon reconnecting or peer window expiration. Logging resumes after the primary database leaves disconnected peer state.

**Note:** If you set **DB2\_HADR\_PEER\_WAIT\_LIMIT**, use a minimum value of 10 to avoid triggering false alarms.

Honoring peer window transition when breaking out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies (safe takeover from standby as long as it is still in disconnected-peer state).

- In most systems, the logging capability is not driven to its limit. Even in SYNC mode, there might not be an observable slow down on the primary database. For example, if the limit of logging is 40 MB per second with HADR enabled, but the system was just running at 30 MB per second before HADR is enabled, then you might not notice any difference in overall system performance.
- To speed up the catchup process, you can use a shared log archive device. However, if the shared device is a serial device such as a tape drive, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations.
- If you are going to use the reads on standby feature, the standby must have the resources to accommodate this additional work.
- If you are going to use the reads on standby feature, configure your buffer pools on the primary, and that information is shipped to the standby through logs.
- If you are going to use the reads on standby feature, tune the **pckcachesz**, **catalogcache\_sz**, **applheapsz**, and **sortheap** configuration parameters on the standby.

## **Cluster managers and high availability disaster recovery (HADR)**

You can implement Db2 high availability disaster recovery (HADR) databases on nodes of a cluster, and use a cluster manager to improve the availability of your database solution.

You can have both the primary database and the standby database managed by the same cluster manager, or you can have the primary database and the standby database managed by different cluster managers.

### **Set up an HADR pair where the primary and standby databases are serviced by the same cluster manager**

This configuration is best suited to environments where the primary and standby databases are located at the same site and where the fastest possible failover is required. These environments would benefit from using HADR to maintain DBMS availability, rather using crash recovery or another recovery method.

You can use the cluster manager to quickly detect a problem and to initiate a takeover operation. Because HADR requires separate storage for the DBMS, the cluster manager should be configured with separate volume control. This configuration prevents the cluster manager from waiting for failover to occur on the volume before using the DBMS on the standby system. You can use the automatic client reroute feature to redirect client applications to the new primary database.

## Set up an HADR pair where the primary and standby databases are not serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at different sites and where high availability is required for disaster recovery in the event of a complete site failure. There are several ways you can implement this configuration. When an HADR primary or standby database is part of a cluster, there are two possible failover scenarios.

- If a partial site failure occurs and a node to which the DBMS can fail over remains available, you can choose to perform a cluster failover. In this case, the IP address and volume failover is performed using the cluster manager; HADR is not affected.
- If a complete site failure occurs where the primary database is located, you can use HADR to maintain DBMS availability by initiating a takeover operation. If a complete site failure occurs where the standby database is located, you can repair the site or move the standby database to another site.

**Note:** For HADR deployments in Db2 pureScale environments, IBM Tivoli System Automation for Multiplatforms cannot be used to automate HADR; SA MP only manages high availability for the local cluster.

### High availability disaster recovery (HADR) synchronization mode

The HADR synchronization mode determines the degree of protection your Db2 high availability disaster recovery (HADR) database solution has against transaction loss. The synchronization mode determines when the primary database server considers a transaction complete, based on the state of the logging on the standby database.

The more strict the synchronization mode configuration parameter value, the more protection your database solution has against transaction data loss, but the slower your transaction processing performance. You must balance the need for protection against transaction loss with the need for performance.

Figure 7 shows the Db2 HADR synchronization modes that are available and also when transactions are considered committed based on the synchronization mode chosen:

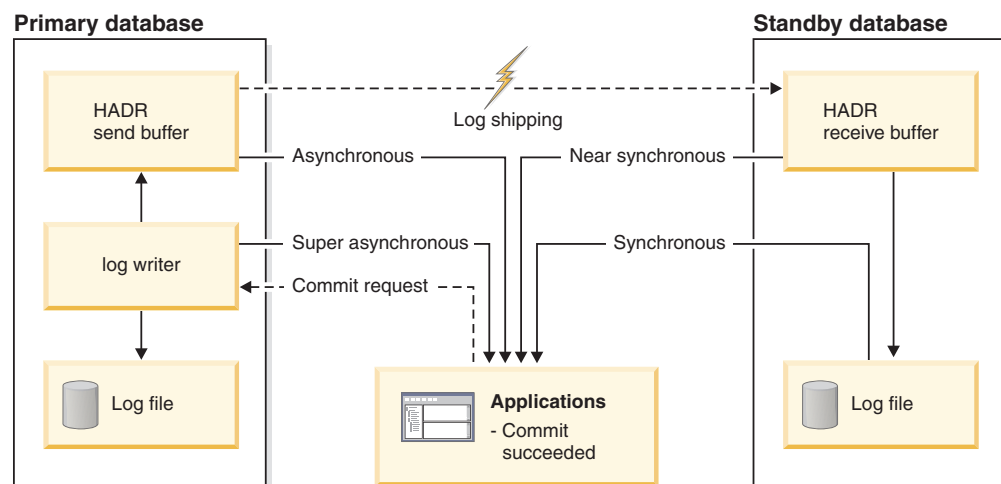


Figure 7. Synchronization modes for high availability and disaster recovery (HADR)

Although you set **hadr\_syncmode** on the primary and the standby databases, the effective synchronization mode is determined by the primary or by the standby's role. That is, auxiliary standbys (any standby that is not listed as the first entry in the primary's target list) automatically have their synchronization modes set to **SUPERASYNC**, and the principal standby (the standby that is listed as the first entry in the primary's target list) uses the synchronization mode set on the primary. A standby's effective synchronization mode is the value that is displayed by any monitoring interface. The only exception to this is when you have not configured the **hadr\_target\_list** parameter. In that case, the primary and standby must have the same setting for **hadr\_syncmode**.

Use the **hadr\_syncmode** database configuration parameter to set the synchronization mode. The following values are valid:

#### **SYNC (synchronous)**

This mode provides the greatest protection against transaction loss, and using it results in the longest transaction response time among the four modes.

In this mode, log writes are considered successful only when logs have been written to log files on the primary database and when the primary database has received acknowledgement from the standby database that the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

If the standby database crashes before it can replay the log records, the next time it starts it can retrieve and replay them from its local log files. If the primary database fails, a failover to the standby database guarantees that any transaction that has been committed on the primary database has also been committed on the standby database. After the failover operation, when the client reconnects to the new primary database, there can be transactions committed on the new primary database that were never reported as committed to the application on the original primary. This occurs when the primary database fails before it processes an acknowledgement message from the standby database. Client applications should consider querying the database to determine whether any such transactions exist.

If the primary database loses its connection to the standby database, what happens next depends on the configuration of the **hadr\_peer\_window** database configuration parameter. If **hadr\_peer\_window** is set to a non-zero time value, then upon losing connection with the standby database the primary database will move into disconnected peer state and continue to wait for acknowledgement from the standby database before committing transactions. If the **hadr\_peer\_window** database configuration parameter is set to zero, the primary and standby databases are no longer considered to be in peer state and transactions will not be held back waiting for acknowledgement from the standby database. If the failover operation is performed when the databases are not in peer or disconnected peer state, there is no guarantee that all of the transactions committed on the primary database will appear on the standby database.

If the primary database fails when the databases are in peer or disconnected peer state, it can rejoin the HADR pair as a standby database after a failover operation. Because a transaction is not considered to be committed until the primary database receives acknowledgement from the standby database that the logs have also been written to log files on the standby database, the log sequence on the primary will be the same as the

log sequence on the standby database. The original primary database (now a standby database) just needs to catch up by replaying the new log records generated on the new primary database since the failover operation.

If the primary database is not in peer state when it fails, its log sequence might be different from the log sequence on the standby database. If a failover operation has to be performed, the log sequence on the primary and standby databases might be different because the standby database starts its own log sequence after the failover. Because some operations cannot be undone (for example, dropping a table), it is not possible to revert the primary database to the point in time when the new log sequence was created. If the log sequences are different and you issue the **START HADR** command with the **AS STANDBY** parameter on the original primary, you will receive a message that the command was successful. However, this message is issued before reintegration is attempted. If reintegration fails, pair validation messages will be issued to the administration log and the diagnostics log on both the primary and the standby. The reintegrated standby will remain the standby, but the primary will reject the standby during pair validation causing the standby database to shut down. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

#### **NEARSYNC (near synchronous)**

While this mode has a shorter transaction response time than synchronous mode, it also provides slightly less protection against transaction loss.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and when the primary database has received acknowledgement from the standby system that the logs have also been written to main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred to nonvolatile storage all of the log data that it has received.

If the standby database crashes before it can copy the log records from memory to disk, the log records will be lost on the standby database. Usually, the standby database can get the missing log records from the primary database when the standby database restarts. However, if a failure on the primary database or the network makes retrieval impossible and a failover is required, the log records will never appear on the standby database, and transactions associated with these log records will never appear on the standby database.

If transactions are lost, the new primary database is not identical to the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records (because both the primary and standby databases have failed), the log sequences on the primary and standby databases will be different and

attempts to restart the original primary database as a standby database without first performing a restore operation will fail. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

#### **ASync (asynchronous)**

Compared with the SYNC and NEARSYNC modes, the ASync mode results in shorter transaction response times but might cause greater transaction losses if the primary database fails

In ASync mode, log writes are considered successful only when the log records have been written to the log files on the primary database and have been delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby database.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater possibility of log records being lost if a failover occurs in asynchronous mode, there is also a greater possibility that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameters. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

**Note:** You cannot set the **hadr\_syncmode** parameter to ASync if you are using peer window functionality (this is, if **hadr\_peer\_window** is set to a nonzero value).

#### **SUPERASync (super asynchronous)**

This mode has the shortest transaction response time but has also the highest probability of transaction losses if the primary system fails. This mode is useful when you do not want transactions to be blocked or experience elongated response times due to network interruptions or congestion.

In this mode, the HADR pair can never be in peer state or disconnected peer state. The log writes are considered successful as soon as the log records have been written to the log files on the primary database. Because the primary database does not wait for acknowledgement from the standby database, transactions are considered committed irrespective of the state of the replication of that transaction.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

Because the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap as it is an indirect measure of the potential number of transactions that might be lost should a true disaster occur on the primary system. In disaster recovery scenarios, any transactions committed during the log gap would not be available to the standby database. Therefore, monitor the log gap by checking the **hadr\_log\_gap** field of the **db2pd** command with the **-hadr** option or the **MON\_GET\_HADR** table function; if the size of the log gap is not acceptable, investigate the network interruptions or the relative speed of the standby database node and take corrective measures to reduce the log gap.

If the primary database fails, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater probability of log records being lost if a failover occurs in super asynchronous mode, there is also a greater probability that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

**Note:** You cannot set the **hadr\_syncmode** parameter to **SUPERASYNC** if you are using peer window functionality (this is, if **hadr\_peer\_window** is set to a nonzero value).

## High availability disaster recovery (HADR) support

To get the most out of the Db2 database High Availability Disaster Recovery (HADR) feature, consider system requirements and feature limitations when designing your high availability database solution.

## **System requirements for Db2 high availability disaster recovery (HADR):**

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following requirements for hardware, operating systems, and for the Db2 database system.

**Recommendation:** For better performance, use the same hardware and software for the system where the primary database resides and for the system where the standby database resides. If the system where the standby database resides has fewer resources than the system where the primary database resides, it is possible that the standby database will be unable to keep up with the transaction load generated by the primary database. This can cause the standby database to fall behind or the performance of the primary database to degrade. In a failover situation, the new primary database should have the resources to service the client applications adequately.

If you enable reads on standby and use the standby database to run some of your read-only workload, ensure that the standby has sufficient resources. An active standby requires additional memory and temporary table space usage to support transactions, sessions, and new threads as well as queries that involve sort and join operations.

### **Hardware and operating system requirements**

**Recommendation:** Use identical host computers for the HADR primary and standby databases. That is, they should be from the same vendor and have the same architecture.

The operating system on the primary and standby databases should be the same version, including patches. When the rolling update procedure is used to upgrade the operating system, the operating system versions can be different on the primary and standby during the procedure. To minimize risks, plan ahead to have the procedure completed in a short time and try it out first in a test environment.

A TCP/IP interface must be available between the HADR host machines, and a high-speed, high-capacity network is recommended.

### **Db2 database requirements**

The versions of the database systems for the primary and standby databases must be identical; for example, both must be either V10.1 or V10.5. During rolling updates, the modification level (for example, the fix pack level) of the database system for the standby database can be later than that of the primary database for a short while to test the new level. However, you should not keep this configuration for an extended period of time. The primary and standby databases will not connect to each other if the modification level of the database system for the primary database is later than that of the standby database. In order to use the reads on standby feature, both the primary and the standby databases need to be Version 9.7 Fix Pack 1.

The Db2 database software for both the primary and standby databases must have the same bit size (32 or 64 bit). Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases.

Storage groups are fully supported by HADR, including replication of the CREATE STOGROUP, ALTER STOGROUP and DROP STOGROUP statements. Similar to table space containers, the storage paths must exist on both primary and standby.

The primary and standby databases must have the same database name. This means that they must be in different instances.

Redirected restore is not supported. That is, HADR does not support redirecting table space containers. However, database directory and log directory changes are supported. Table space containers created by relative paths will be restored to paths relative to the new database directory.

### **Buffer pool requirements**

Since buffer pool operations are also replayed on the standby database, it is important that the primary and standby databases have the same amount of memory. If you are using reads on standby, you will need to configure the buffer pool on the primary so that the active standby can accommodate log replay and read applications.

### **Installation and storage requirements for high availability disaster recovery (HADR):**

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following installation and storage requirements.

#### **Installation requirements**

For HADR, instance paths should be the same on the primary and the standby databases. Using different instance paths can cause problems in some situations, such as if an SQL stored procedure invokes a user-defined function (UDF) and the path to the UDF object code is expected to be on the same directory for both the primary and standby server.

#### **Storage requirements**

Storage groups are fully supported by HADR, including replication of the CREATE STOGROUP, ALTER STOGROUP and DROP STOGROUP statements. Similar to table space containers, the storage path must exist on both primary and standby. Symbolic links can be used to create identical paths. The primary and standby databases can be on the same computer. Even though their database storage starts at the same path, they do not conflict because the actual directories used have instance names embedded in them (since the primary and standby databases must have the same database name, they must be in different instances). The storage path is formulated as `storage_path_name/inst_name/dbpart_name/db_name/tbsp_name/container_name`.



Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include: the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). Storage groups and their storage paths must be identical. This includes the path names and the amount of space on each that is devoted to each storage group. The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If the table space setup is not identical on the primary and standby databases, log replay on the standby database might encounter errors such as OUT OF SPACE or TABLE SPACE CONTAINER NOT FOUND. Similarly, if the storage groups's storage path setup is not identical on the primary and standby databases, log records associated with the CREATE STOGROUP or ALTER STOGROUP are not be replayed. As a result, the existing storage paths might prematurely run out of space on the standby system and automatic storage table spaces are not be able to increase in size. If any of these situations occurs, the affected table space is put in rollforward pending state and is ignored in subsequent log replay. If a takeover operation occurs, the table space is not available to applications.

If the problem is noticed on the standby system prior to a takeover then the resolution is to re-establish the standby database while addressing the storage issues. The steps to do this include:

- Deactivating the standby database.
- Dropping the standby database.
- Ensuring the necessary file systems exist with enough free space for the subsequent restore and rollforward.
- Restoring the database at the standby system using a recent backup of the primary database (or, reinitialize using split mirror or flash copy with the **db2inidb** command). Storage group storage paths should not be redefined during the restore. Also, table space containers should not be redirected as part of the restore.
- Restarting HADR on the standby system.

However, if the problem is noticed with the standby database after a takeover has occurred (or if a choice was made to not address the storage issues until this time) then the resolution is based on the type of problem that was encountered.

If the database is enabled for automatic storage and space is not available on the storage paths associated with the standby database then follow these steps:

1. Make space available on the storage paths by extending the file systems, or by removing unnecessary non-Db2 files on them.
2. Perform a table space rollforward to the end of logs.

In the case where the addition or extension of containers as part of log replay could not occur, if the necessary backup images and log file archives are available, you might be able to recover the table space by first issuing the SET TABLESPACE CONTAINERS statement with the IGNORE ROLLFORWARD CONTAINER OPERATIONS option and then issuing the **ROLLFORWARD** command.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases. Consequently, if the primary and standby databases are placed on the same computer, all table space containers must be defined with relative paths so that they map to different paths for primary and standby.

### **HADR and network address translation (NAT) support:**

Network address translation (NAT) is usually used for firewall and security because it hides the server's real address. NAT is supported in HADR environments unless you are also using the Db2 pureScale Feature.

In an HADR setup, the local and remote host configurations on the primary and standby nodes are cross-checked to ensure that they are correct. In a NAT environment, a host is known to itself by a particular IP address but is known to the other hosts by a different IP address. This behavior causes the HADR host cross-check to fail unless you set the **DB2\_HADR\_NO\_IP\_CHECK** registry variable to ON. Using this setting causes the host cross-check to be bypassed, enabling the primary and standby to connect in a NAT environment.

If you are not running in a NAT environment, use the default setting of OFF for the **DB2\_HADR\_NO\_IP\_CHECK** registry variable. Disabling the cross-check weakens the HADR validation of your configuration.

### **Multiple HADR standby databases**

Normally, with multiple standby databases, on startup, a standby checks that its settings for the **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters are also used for its **hadr\_target\_list** parameter. This check is done to ensure that on role switch, the old primary can become a new standby. In NAT scenarios, that check fails unless you set the **DB2\_HADR\_NO\_IP\_CHECK** registry variable to ON. Because this check is bypassed when **DB2\_HADR\_NO\_IP\_CHECK** is set to ON, the standby waits until it connects to the primary to check that the values of the primary's **hadr\_local\_host** and **hadr\_local\_svc** configuration parameters are used for the standby's **hadr\_target\_list** configuration parameter. The check still ensures that role switch can succeed for the standby and primary pair.

**Important:** If you set the **DB2\_HADR\_NO\_IP\_CHECK** registry variable to ON, the values of the **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters are not automatically updated.

In a multiple standby setup, you should set the **DB2\_HADR\_NO\_IP\_CHECK** registry variable for all databases that might connect to another database across a NAT boundary. If a database will never cross a NAT boundary to connect to another database (that is, if no such link is configured), you should not set this registry variable for that database. If you set the **DB2\_HADR\_NO\_IP\_CHECK** registry variable, it prevents a standby from automatically discovering the new primary after a takeover has occurred, and you must manually reconfigure the standby to have it connect to the new primary.

### **Restrictions for high availability disaster recovery (HADR):**

To help achieve optimal performance with high availability disaster recovery (HADR), consider HADR restrictions when designing your high availability Db2 database solution.

HADR restrictions are as follows:

- HADR is not supported in a partitioned database environment.
- The primary and standby databases must be on the same operating system version and must use the same level of the Db2 database system, except for a short time during a rolling upgrade.
- The Db2 software that you use for the primary database and the Db2 software that you use for the standby databases must be the same bit size (32 or 64 bit).
- Clients cannot connect to the standby database unless you enable the reads on standby feature. This feature enables clients to connect to the active standby database and issue read-only queries.
- Only read clients can connect to an active standby database; however, operations on the standby database that write a log record are not permitted, nor are the following operations that modify database contents:
  - any asynchronous threads such as real-time statistics collection
  - automatic index rebuilds and utilities that modify database objects
- Log files are archived only by the primary database.
- You can run the self-tuning memory manager (STMM) only on the current primary database. After you start the primary database or convert the standby database to a primary database by takeover, the STMM EDU might not start until the first client connection is made.
- Backup operations are not supported on the standby database.
- The **SET WRITE** command cannot be issued on the standby database.
- Non-logged operations, such as changes to database configuration parameters, the recovery history file, and LOB table columns for which you specified the NOT LOGGED parameter, are not replicated to the standby database.
- Load operations for which you specify the **COPY NO** parameter are not supported.
- HADR does not support the use of raw I/O (direct disk access) for database log files. If you start HADR by using the **START HADR** command or the database is activated or restarted with HADR configured and raw logs are detected, the associated command fails.
- Federated servers do not fully support HADR in federated two-phase commit (F2PC) scenarios. If you configure an HADR database as a federated database, it supports F2PC only with type-1 inbound connections.
- HADR does not support infinite logging.
- Ensure that the system clock of the HADR primary database is synchronized with the system clock of the HADR standby database.

## Scheduling maintenance for high availability

Your Db2 database solution will require regular maintenance. You will have to perform maintenance such as: software or hardware upgrades; database performance tuning; database backups; statistics collection and monitoring for business purposes. You must minimize the impact of these maintenance activities on the availability of your database solution.

### Before you begin

Before you can schedule maintenance activities, you must identify those maintenance activities that you will have to perform on your database solution.

## Procedure

To schedule maintenance, perform the following steps:

1. Identify periods of low database activity.  
It is best to schedule maintenance activities for low-usage times (those periods of time when the fewest user applications are making requests of the database system). Depending on the type of business applications you are creating, there might even be periods of time when no user applications are accessing the database system.
2. Categorize the maintenance activities you must perform according to the following:
  - The maintenance can be automated
  - You must bring the database solution offline while you perform the maintenance
  - You can perform the maintenance while the database solution is online
3. For those maintenance activities that can be automated, configure automated maintenance using one of the following methods:
  - Use the **auto\_maint** configuration parameter
  - Use one of the system stored procedure called AUTOMAINT\_SET\_POLICY and AUTOMAINT\_SET\_POLICYFILE
4. If any of the maintenance activities you must perform require the database server to be offline, schedule those offline maintenance activities for those low-usage times.
5. For those maintenance activities that can be performed while the database server is online:
  - Identify the availability impact of running those online maintenance activities.
  - Schedule those online maintenance activities so as to minimize the impact of running those maintenance activities on the availability of the database system.

For example: schedule online maintenance activities for low-usage times; and use throttling mechanisms to balance the amount of system resources the maintenance activities use.

### **Configuring an automated maintenance policy using SYSPROC.AUTOMAINT\_SET\_POLICY or SYSPROC.AUTOMAINT\_SET\_POLICYFILE**

You can use the system stored procedures AUTOMAINT\_SET\_POLICY and AUTOMAINT\_SET\_POLICYFILE to configure the automated maintenance policy for a database.

## Procedure

To configure the automated maintenance policy for a database, perform the following steps:

1. Connect to the database
2. Call AUTOMAINT\_SET\_POLICY or AUTOMAINT\_SET\_POLICYFILE
  - The parameters required for AUTOMAINT\_SET\_POLICY are:
    - a. Maintenance type, specifying the type of automated maintenance activity to configure.

- b. Pointer to a BLOB that specifies the automated maintenance policy in XML format.
- The parameters required for AUTOMAINT\_SET\_POLICYFILE are:
  - a. Maintenance type, specifying the type of automated maintenance activity to configure.
  - b. The name of an XML file that specifies the automated maintenance policy.

Valid maintenance type values are:

- AUTO\_BACKUP - automatic backup
- AUTO\_REORG - automatic table and index reorganization
- AUTO\_RUNSTATS - automatic table **RUNSTATS** operations
- MAINTENANCE\_WINDOW - maintenance window

## What to do next

You can use the system stored procedures AUTOMAINT\_GET\_POLICY and AUTOMAINT\_GET\_POLICYFILE to retrieve the automated maintenance policy configured for a database.

### Sample automated maintenance policy specification XML for AUTOMAINT\_SET\_POLICY or AUTOMAINT\_SET\_POLICYFILE:

Whether you are using AUTOMAINT\_SET\_POLICY or AUTOMAINT\_SET\_POLICYFILE to specify your automated maintenance policy, you must specify the policy using XML. There are sample files that demonstrate how to specify your automated maintenance policy in XML. In Linux and UNIX operating systems, you can find the sample files in the \$QLLIB/samples/automaintcfg directory. In Windows operating systems, you can find the sample files in the \$QLLIB\samples\automaintcfg directory.

The second parameter you pass to the system stored procedure AUTOMAINT\_SET\_POLICY is a BLOB containing XML, specifying your desired automated maintenance policy. The second parameter you pass to the system stored procedure AUTOMAINT\_SET\_POLICYFILE is the name of an XML file that specifies your desired automated maintenance policy. The XML elements that are valid in the BLOB you pass to AUTOMAINT\_SET\_POLICY are the same elements that are valid in the XML file you pass to AUTOMAINT\_SET\_POLICYFILE.

In the samples directory (\$QLLIB/samples/automaintcfg in Linux and UNIX environments and \$QLLIB\samples\automaintcfg in Windows environments) there are four XML files that contain example automated maintenance policy specification:

#### **DB2MaintenanceWindowPolicySample.xml**

Demonstrates specifying a maintenance window during which time the database manager should schedule automated maintenance.

#### **DB2AutoBackupPolicySample.xml**

Demonstrates specifying how the database manager should perform automatic backup.

#### **DB2AutoReorgPolicySample.xml**

Demonstrates specifying how the database manager should perform automatic table and index reorganization.

## DB2DefaultAutoRunstatsPolicySample.xml

Demonstrates specifying how the database manager should perform automatic table **runstats** operations.

You can create your own automated maintenance policy specification XML by copying the XML from these files and modifying that XML according to the requirements of your system.

## Configuring database logging options

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

### Before you begin

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

### About this task

You can configure database logging options by using the **UPDATE DATABASE CONFIGURATION** command on the command line processor (CLP), or by calling the db2CfgSet API.

### Procedure

- To configure database logging options by using the **UPDATE DATABASE CONFIGURATION** command on the command line processor:
  1. Specify whether you want to use circular logging or archive logging. If you want to use circular logging, the **logarchmeth1** and **logarchmeth2** database configuration parameters must be set to OFF. This setting is the default. To use archive logging, you must set at least one of these database configuration parameters to a value other than OFF. For example, if you want to use archive logging and you want to save the archived logs to disk, issue the following command:

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

The archived logs are placed in a directory called /u/dbuser/archived\_logs.

2. Specify values for other database logging configuration parameters, as required. The following additional configuration parameters apply to database logging:
  - **archretrydelay**
  - **blk\_log\_dsk\_ful**
  - **failarchpath**
  - **logarchcompr1**
  - **logarchcompr2**
  - **logarchmeth1**
  - **logarchmeth2**
  - **logarchopt1**
  - **logarchopt2**
  - **logbufsz**

- **logfilsiz**
- **logprimary**
- **logsecond**
- **max\_log**
- **mirrorlogpath**
- **newlogpath**
- **mincommit**
- **numarchretry**
- **num\_log\_span**
- **overflowlogpath**

For more information about these database logging configuration parameters, see “Configuration parameters for database logging.”

- To configure database logging options by using IBM Data Studio, use the task assistant for the **UPDATE DATABASE CONFIGURATION** command.

## Configuration parameters for database logging

A key element of any high availability strategy is database logging. You can use database logs to record transaction information, synchronize primary and secondary (standby) databases, and roll forward a secondary database that has taken over for a failed primary database.

To configure these database logging activities, you must set a variety of database configuration parameters.

### Archive retry delay (archretrydelay)

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

### Block on log disk full (blk\_log\_dsk\_ful)

This configuration parameter can be set to prevent disk full errors from being generated when the Db2 database manager cannot create a new log file in the active log path. Instead, the Db2 database manager will attempt to create the log file every five minutes until it succeeds. After each attempt, the Db2 database manager will write a message to the administration notification log. The only way to confirm that your application is hanging because of a log disk full condition is to monitor the administration notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries might not be directly affected; however, if a query needs to access data that is locked by an update request or a data page that is fixed in the buffer pool by the updating application, read-only queries will also be blocked.

Setting **blk\_log\_dsk\_ful** to YES causes applications to hang when the Db2 database manager encounters a log disk full error. You are then able to resolve the error and the application can continue. A disk full situation can be resolved by moving old log files to another file system, by increasing the size of the file system so that hanging applications can complete, or by investigating and resolving any log archiving failures.

If **blk\_log\_dsk\_ful** is set to NO, a transaction that receives a log disk full error will fail and be rolled back.

### Failover archive path (failarchpath)

Specifies an alternate directory for the archive log files if there is a problem

with the normal archive path (for example, if it is not accessible or full). This directory is a temporary storage area for the log files until the log archive method that failed becomes available again, at which time the log files will be moved from this directory to the path specified in the original log archiving. Moving the log files to this temporary location, helps you avoid log directory full situations. This parameter must be a fully qualified existing directory.

#### **Primary log archive compression (logarchcompr1), secondary log archive compression (logarchcompr2)**

In certain circumstances, these parameters control whether the database manager compresses archive log files. You can reduce the cost associated with storing log archive files if you use compression on the files.

Valid values for these parameters are as follows:

- OFF** This value specifies that log archive files are not compressed. The default value is OFF.
- ON** This value specifies that log archive files are compressed. If set dynamically, log files already archived are not compressed.

#### **Note:**

1. If you set the **logarchmeth1** configuration parameter to a value other than DISK, TSM, or VENDOR, log archive compression has no effect regardless of the **logarchcompr1** configuration parameter setting.
2. If you set the **logarchmeth2** configuration parameter to a value other than DISK, TSM, or VENDOR, log archive compression has no effect regardless of the **logarchcompr2** configuration parameter setting.

#### **Log archive method 1 (logarchmeth1), log archive method 2 (logarchmeth2)**

These parameters cause the database manager to archive log files to a location that is not the active log path. If you specify both of these parameters, each log file from the active log path that is set by the **logpath** configuration parameter is archived twice. This means that you will have two identical copies of archived log files from the log path in two different destinations. If you specify mirror logging by using the **mirrorlogpath** configuration parameter, the **logarchmeth2** configuration parameter archives log files from the mirror log path instead of archiving additional copies of the log files in the active log path. This means that you have two separate copies of the log files archived in two different destinations: one copy from the log path and one copy from the mirror log path.

Valid values for these parameters are as follows:

- OFF** This value specifies that the log archiving method is not used. If you set both the **logarchmeth1** and **logarchmeth2** configuration parameters to OFF, the database is considered to be using circular logging and is not rollforward recoverable. The default value is OFF.

#### **LOGRETAIN**

Specifies that active log files are retained and become online archive log files for use in rollforward recovery.

#### **USEREXIT**

Specifies that log retention logging is performed and that a user exit program should be used to archive and retrieve the log files. Log files are archived when they are full. They are retrieved when the rollforward utility must use them to restore a database.



**DISK** You must follow this value with a colon (:) and then a fully qualified existing path name where the log files will be archived. For example, if you set the **logarchmeth1** configuration parameter to **DISK:/u/dbuser/archived\_logs**, the archive log files are placed under or in the **/u/dbuser/archived\_logs/INSTANCE\_NAME/DBNAME/NODExxxx/LOGSTREAMxxxx/Cxxxxxxx** directory.

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

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

In addition, see the *Log archive options 1 (logarchopt1)*, *log archive options 2 (logarchopt2)* section below for configurable options that control Db2 and TSM log archive behavior.

#### **VENDOR**

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

In addition, see the *Log archive options 1 (logarchopt1)*, *log archive options 2 (logarchopt2)* section below for configurable options that control Db2 and vendor log archive behavior.

#### **Note:**

1. If either **logarchmeth1** or **logarchmeth2** is set to a value other than OFF, the database is configured for rollforward recovery.

#### **Log archive options 1 (logarchopt1), log archive options 2 (logarchopt2)**

Specifies a string of options which control log archiving behavior when a log archive method (**logarchmeth1**, **logarchmeth2**, or both) is configured. Multiple options can be specified in the string and must be separated by white space.

There are two classes of options, options that are recognized by Db2 and influence the database manager's log archiving behavior, and options that are not recognized by Db2 and passed directly into the TSM or vendor APIs to influence vendor or storage manager behavior.

#### **Options that are recognized by Db2 and influence database manager log archiving behavior:**

The following options influence Db2 log archiving database behavior. These options must be preceded by 2 hyphen or dash characters "--", and must be the first entry or entries in the string.

#### **"--DBNAME"**

For TSM and VENDOR log archive methods, use this parameter to enable the database to retrieve log files that were generated using a different database name. This is useful when a database backup image is restored into a new database name (by using the restore REDIRECT option) and a subsequent database rollforward operation must retrieve log files from the archive using the original database name. For more details, see Technote #1687492 DB2 fails

to retrieve archived log after Redirected Restore into a new database name. An example is illustrated below:

```
"--dbname=<original database name>"
```

### **"--VENDOR\_ARCHIVE\_TIMEOUT" (Unix platforms only)**

For TSM and VENDOR log archive methods, starting in Db2 Version 11.1.3.3, use this parameter to enforce a timeout (in seconds) when the database manager attempts to archive a log file. If the transmission of log data between Db2 and the TSM or vendor API is unresponsive for the specified timeout period, then Db2 will interrupt the archive log attempt and follow normal log archive failure protocol (if a failover log archive path (FAILARCHPATH) is configured then Db2 will attempt to archive the log file to this path, otherwise Db2 will retry the archive log attempt).

This option is supported on UNIX platforms only.

Values in the range 1 - 2,147,483,674 are permitted.

When the **logarchopt1** or **logarchopt2** database configuration parameter is updated to include this option, a database instance restart is not required, however the timeout will be enforced at the start of the next archive log attempt. Monitoring of log archiving to TSM or Vendor methods can be accomplished by using the **db2pd -fvp** command. For examples, of **db2pd -fvp** with log archive requests see: db2pd - Monitor and troubleshoot DB2 database command. An example is illustrated below:

```
"--vendor_archive_timeout=<number of seconds>"
```

### **Options that are passed directly on to the TSM or vendor APIs:**

Any remaining options in the string will not be recognized by Db2, and the string (excluding any options that were recognized by Db2 as described above) is passed directly to the TSM or vendor APIs.

For TSM environments, some commonly used TSM options for Db2 database environments include:

TSM requires that options are preceded by one hyphen/dash character "-". Specifies a string of options that control log archiving behavior when a log archive method (**logarchmeth1**, **logarchmeth2** or both) is configured. Multiple options can be specified in the string and must be separated by white space.

To enable the database to retrieve logs that were generated on a different TSM node, by a different TSM user, or in TSM environments that use proxy nodes such as in Db2 pureScale environments, you must provide the string in one of the following formats:

- For retrieving logs generated on a different TSM node when the TSM server is not configured to support proxy node clients:  
"--fromnode=nodename"
- For retrieving logs generated by a different TSM user when the TSM server is not configured to support proxy node clients:  
"--fromowner=ownername"
- For retrieving logs generated on a different TSM node and by a different TSM user when the TSM server is not configured to support proxy node clients:

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

- For retrieving logs generated in client proxy nodes configurations, such as in Db2 pureScale environments where there are multiples members working on the same data:

`"-asnodename=proxynode"`

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

#### Restrictions:

- When the `-asnodename` TSM option is used, data is not stored using the name of the node (*nodename*) of each member. The data is stored instead using the name of the shared TSM target node used by all the members within a Db2 pureScale instance.
- The `-fromnode` option and the `-fromowner` option are not compatible with the `-asnodename` option and cannot be used together. Use the `-asnodename` option for TSM configurations using proxy nodes and the other two options for other types of TSM configurations. For more information, see “Configuring a Tivoli Storage Manager client” on page 436.

#### Log buffer (logbufsz)

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

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

Increasing the log buffer size can result in more efficient input/output (I/O) activity associated with logging, because the log records are written to disk less frequently, and more records are written each time. However, recovery can take longer with a larger log buffer size value. As well, you may be able to use a higher **logbufsz** setting to reduce number of reads from the log disk. (To determine if your system would benefit from this, use the **log\_reads** monitor element to check if reading from log disk is significant.

#### Log file size (logfilsiz)

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

There is a 1024 GB logical limit on the total active log space per log stream that you can configure. This limit is the result of the upper limit for each log file, which is 4 GB, and the maximum combined number of primary and secondary log files, which is 256.

The size of the log file has a direct bearing on performance. There is a performance cost for switching from one log to another. So, from a pure performance perspective, the larger the log file size the better. This parameter also indicates the log file size for archiving. In this case, a larger log file is size it not necessarily better, since a larger log file size can increase the chance of failure or cause a delay in log shipping scenarios. When considering active log space, it might be better to have a larger

number of smaller log files. For example, if there are two very large log files and a transaction starts close to the end of one log file, only half of the log space remains available.

Every time a database is deactivated (all connections to the database are terminated), the log file that is currently being written is truncated. So, if a database is frequently being deactivated, it is better not to choose a large log file size because the Db2 database manager will create a large file only to have it truncated. You can use the **ACTIVATE DATABASE** command to avoid this cost because it prevents automatic database deactivation when the last client disconnects from the database.

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

Minimizing log file loss is also an important consideration when setting the log size. Archiving operates on one entire log file at a time. If you configure larger log files, you increase the time between archiving. If the medium containing the log fails, some transaction information will probably be lost. Decreasing the log file size increases the frequency of archiving but can reduce the amount of information loss in case of a media failure because on average less log data is not yet archived at any given point in time.

#### **Maximum log per transaction (max\_log)**

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

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

You can override this behavior by setting the **DB2\_FORCE\_APP\_ON\_MAX\_LOG** registry variable to FALSE. This will cause transactions that violate the **max\_log** configuration to fail. The application can still commit the work completed by previous statements in the unit of work, or it can roll back the completed work to undo the unit of work.

This parameter, along with the **num\_log\_span** configuration parameter, can be useful when infinite active log space is enabled. If infinite logging is on (that is, if **logsecond** is -1) then transactions are not restricted to the upper limit of the number of log files (**logprimary** + **logsecond**). When the value of **logprimary** is reached, the Db2 database manager starts to archive the active logs, rather than failing the transaction. This can cause problems if, for instance, there is a long running transaction that has been left uncommitted (perhaps caused by an application with a logic error). If this occurs, the active log space keeps growing, which might lead to poor crash recovery performance. To prevent this, you can specify values for either one or both of the **max\_log** and **num\_log\_span** configuration parameters.

**Note:** The following Db2 commands are excluded from the limitation imposed by the **max\_log** configuration parameter: **ARCHIVE LOG**, **BACKUP DATABASE**, **LOAD**, **REORG**, **RESTORE DATABASE**, and **ROLLFORWARD DATABASE**.

#### **Mirror log path (mirrorlogpath)**

To protect the logs on the primary log path from disk failure or accidental

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

The **mirrorlogpath** parameter also has an effect on log archiving behavior, which you can use to further improve resilience during rollforward recovery: When both **mirrorlogpath** and **logarchmeth2** are set, **logarchmeth2** archives log files from the mirror log path instead of archiving additional copies of the log files in the active log path. You can use this log archiving behaviour to improve resilience, because a usable, archived log file from the mirror log path might still be available to continue a database recovery operation, even if a primary log file became corrupted due to a disk failure before archiving.

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

You must keep track of the location of the logs.

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

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

**Note:**

1. This configuration parameter is not supported if the primary log path is a raw device.
2. The value specified for this parameter cannot be a raw device.
3. In a Db2 pureScale environment, the first member connecting to or activating the database processes configuration changes to this log path parameter. The Db2 database manager verifies that the path exists and that it has both read and write access to that path. It also creates member-specific subdirectories for the log files. If any one of these operations fails, the Db2 database manager rejects the specified path and brings the database online using the old path. If the specified path is accepted, the new value is propagated to each member. If a member fails while trying to switch to the new path, subsequent attempts to activate it or to connect to it will fail (SQL5099N). All members must use the same log path.

**New log path (newlogpath)**

The database logs are initially created in the following directory: *db\_path/instance\_name/dbname/NODE0000/LOGSTREAM0000*. You can change the location in which active log files are placed (and future log files will be placed) by changing the value of this configuration parameter to point to a different directory or to a device. Active logs that are currently stored in the database log path directory are not moved to the new location if the database is configured for rollforward recovery.

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

You must keep track of the location of the logs.

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

**Note:** In a Db2 pureScale environment, the first member connecting to or activating the database processes configuration changes to this log path parameter. The Db2 database manager verifies that the path exists and that it has both read and write access to that path. It also creates member-specific subdirectories for the log files. If any one of these operations fails, the Db2 database manager rejects the specified path and brings the database online using the old path. If the specified path is accepted, the new value is propagated to each member. If a member fails while trying to switch to the new path, subsequent attempts to activate it or to connect to it will fail (SQL5099N). All members must use the same log path.

#### **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 multiple applications attempt to commit their transactions at about the same time. When commit grouping is in effect, application commit requests are held until either one second has elapsed, or the number of commit requests equals the value of this parameter.

#### **Number of archive retries on error (numarchretry)**

Specifies the number of attempts that will be made to archive log files using a configured log archive method before they are archived to the path specified by the **failarchpath** configuration parameter. This parameter can only be used if the **failarchpath** configuration parameter is set. The default value is 5.

#### **Number of active logs a transaction can span (num\_log\_span)**

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

If an application violates the **num\_log\_span** setting, the application will be forced to disconnect from the database.

This parameter, along with the **max\_log** configuration parameter, can be useful when infinite active log space is enabled. If infinite logging is on (that is, if **logsecond** is -1) then transactions are not restricted to the upper limit of the number of log files (**logprimary** + **logsecond**). When the value of **logprimary** is reached, the Db2 database manager starts to archive the active logs, rather than failing the transaction. This can cause problems if, for instance, there is a long running transaction that has been left uncommitted (perhaps caused by an application with a logic error). If this occurs, the active log space keeps growing, which might lead to poor crash recovery performance. To prevent this, you can specify values for either one or both of the **max\_log** and **num\_log\_span** configuration parameters.

**Note:** The following Db2 commands are excluded from the limitation imposed by the **num\_log\_span** configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG, RESTORE DATABASE, and ROLLFORWARD DATABASE.

### Overflow log path (overflowlogpath)

This parameter can be used for several functions, depending on your logging requirements. You can specify a location for the Db2 database manager to find log files that are needed for a rollforward operation. It is similar to the OVERFLOW LOG PATH option of the ROLLFORWARD command; however, instead of specifying the OVERFLOW LOG PATH option for every ROLLFORWARD command issued, you can set this configuration parameter once. If both are used, the OVERFLOW LOG PATH option will overwrite the **overflowlogpath** configuration parameter for that rollforward operation.

If **logsecond** is set to -1, you can specify a directory for the Db2 database manager to store active log files retrieved from the archive. (Active log files must be retrieved for rollback operations if they are no longer in the active log path).

If **overflowlogpath** is not specified, the Db2 database manager will retrieve the log files into the active log path. By specifying this parameter you can provide an additional storage resource where the Db2 database manager can place the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.

For example, if you are using the db2ReadLog API for replication, you can use **overflowlogpath** to specify a location for the Db2 database manager to search for log files that are needed for this API. If the log file is not found (in either the active log path or the overflow log path) and the database is configured for log archiving, the Db2 database manager will retrieve the log file. You can also use this parameter to specify a directory for the Db2 database manager to store the retrieved log files. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.

Setting **overflowlogpath** is useful when infinite logging is configured (i.e., when **logsecond** is set to -1). The Db2 database manager can store active log files retrieved from the archive in this path. (With infinite logging, active log files may need to be retrieved from archive, for rollback or crash recovery operations, if they are no longer in the active log path.)

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

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

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

### Primary log files (logprimary)

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

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

If you are enabling an existing database for rollforward recovery, change the number of primary logs to the sum of the number of primary and secondary logs, plus one.

### **Secondary logs (logsecond)**

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

If the primary log files become full, secondary log files (of size **logfilsiz**) are allocated, one at a time as needed, up to the maximum number specified by this parameter. If this parameter is set to -1, the database is configured with infinite active log space. There is no limit on the size or number of in-flight transactions running on the database. Infinite active logging is useful in environments that must accommodate large jobs requiring more log space than you would normally allocate to the primary logs.

#### **Note:**

1. Log archiving must be enabled in order to set **logsecond** to -1.
2. If this parameter is set to -1, crash recovery time might be increased since the Db2 database manager might need to retrieve archived log files.

### **Reducing logging with the NOT LOGGED INITIALLY parameter**

If your application creates and populates work tables from master tables, you can create the work tables and specify the NOT LOGGED INITIALLY parameter on the CREATE TABLE statement.

This option is useful if you are not concerned about the recoverability of these work tables because they can be easily re-created from the master tables. Specifying the NOT LOGGED INITIALLY parameter reduces logging and improves performance.

The advantage of using the NOT LOGGED INITIALLY parameter is that any changes made on a table (including insert, delete, update, or create index operations) in the same unit of work that creates the table will not be logged. This not only reduces the logging that is done, but can also increase the performance of your application. You can achieve the same result for existing tables by using the ALTER TABLE statement with the NOT LOGGED INITIALLY parameter.

#### **Note:**

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

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



- All changes to the table will be flushed out to disk at commit time. This means that the commit might take longer.
- If the NOT LOGGED INITIALLY attribute is activated and an activity occurs that is not logged, the entire unit of work will be rolled back if a statement fails or a ROLLBACK TO SAVEPOINT is executed (SQL1476N).
- If you are using high availability disaster recovery (HADR) you should not use the NOT LOGGED INITIALLY table attribute. Tables created on the primary database with the NOT LOGGED INITIALLY option specified are not replicated to the standby database. Attempts to access such tables on an active standby database or after the standby becomes the primary as a result of a takeover operation will result in an error (SQL1477N).
- You cannot recover these tables when rolling forward. If the rollforward operation encounters a table that was created or altered with the NOT LOGGED INITIALLY option, the table is marked as unavailable. After the database is recovered, any attempt to access the table returns SQL1477N.

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

### Reducing logging with declared temporary tables

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

- Declared temporary tables are not created in the catalogs; therefore locks are not held.
- Logging is not performed against declared temporary tables, even after the first COMMIT.
- Use the ON COMMIT PRESERVE option to keep the rows in the table after a COMMIT; otherwise, all rows will be deleted.
- Only the application that creates the declared temporary table can access that instance of the table.
- The table is implicitly dropped when the application connection to the database is dropped.
- Created temporary tables (CGTTs) and declared temporary tables (DGTts) cannot be created or accessed on an active standby.
- 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).

### Blocking transactions when the log directory is full

When the Db2 database manager cannot create a log file in the active log path because there is not enough room for the new file, you get errors indicating the disk is full.

If you set the **blk\_log\_dsk\_ful** database configuration parameter, the Db2 database manager repeatedly attempts to create the log file until the file is successfully created instead of returning “disk full” errors.

If you set the **blk\_log\_dsk\_ful** database configuration parameter, the Db2 database manager attempts to create the log file every 5 minutes until it succeeds. If a log

archiving method is specified, the Db2 database manager also checks for the completion of log file archiving. If an archived log file is archived successfully, the Db2 database manager can rename the inactive log file to the new log file name and continue. After each attempt, the Db2 database manager writes a message to the administration notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the administration notification log.

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

### Log file management through log archiving

Db2 server log file archiving is complicated by various operating-system file handling and scheduling problems. For example, if a disk fails as the Db2 database manager is archiving a queue of log files, those log files and the transaction data that they contain might be lost.

Correctly configuring database logging can prevent these kinds of problems from undermining your availability and recovery strategy.

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

- The **logarchcompr1** database configuration parameter specifies whether the database manager compresses log files that are contained in the location specified by **logarchmeth1**. If the **logarchmeth1** configuration parameter is a value other than DISK, TSM, or VENDOR, log archive compression has no effect regardless of the **logarchcompr1** configuration parameter setting.
- The **logarchcompr2** database configuration parameter specifies whether the database manager compresses log files that are contained in the location specified by **logarchmeth2**. If the **logarchmeth2** configuration parameter is a value other than DISK, TSM, or VENDOR, log archive compression has no effect regardless of the **logarchcompr2** configuration parameter setting.
- The **logarchmeth1** database configuration parameter causes the database manager to archive log files or to retrieve log files during rollforward recovery of databases by using the method that you specify. 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. Log files are archived from the path that is specified by the **logpath** configuration parameter.

The **logarchmeth2** database configuration parameter causes the database manager to archive additional copies of log files. If you configure mirror logging, the log files that are archived to the path that is specified by the **logarchmeth2** parameter are taken from the mirror log path. If you do not configure mirror logging, the log files that are archived to the path that is specified by the **logarchmeth2** parameter are taken from the current log path.

- You should not use locally attached tape drives to store log files if you are using any of the following features:
  - Infinite logging
  - Online recovery at the table space level
  - Replication
  - db2ReadLog API
  - High availability disaster recovery (HADR)

Any of these features can cause a log file to be retrieved, which can conflict with log archiving operations. Also, you cannot use locally attached tape drives in a Db2 pureScale environment because the member that is performing the log merge operation must retrieve logs for the other members.

- If you are using log archiving, the log manager attempts to archive active logs as they are filled. In some cases, if a database is deactivated before the log manager can record the archive as successful, the log manager might try to archive the log again when the database is activated. Thus, a log file can be archived more than once.
- If you use archiving, a log file is passed to the log manager when it is full, even if the log file is still active and is needed for normal processing. This process allows copies of the data to be moved away from volatile media as quickly as possible. The log file that is passed to the log manager is retained in the log path directory until it is no longer needed for normal processing. At this point, the disk space is reused.
- If a log file is archived and contains no open transactions, the Db2 database manager does not delete the file but renames it as the next log file when such a file is needed. This process improves performance because creating a new log file instead of renaming the file would require all pages to be written out to guarantee that the necessary disk space or other storage space is available. The database manager retains up to 8 extra log files in the active log path for renaming purposes.
- During crash recovery, during member crash recovery (in a Db2 pureScale environment), or during runtime rollback, the Db2 database manager does not retrieve log files unless you set the **logsecond** database configuration parameter to -1 (that is, if you enable infinite logging). In a Db2 pureScale environment, the database manager might have to retrieve archived logs during a group crash recovery even if you do not enable infinite logging.
- Configuring log archiving does not guarantee rollforward recovery to the point of failure but only attempts to make the failure window smaller. As log files are filled, the log manager asynchronously archives the logs. If the disk that contains the log fails before a log file is filled, the data in that log file is lost. Also, because 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.

To help prevent the case where a failure of the disk or device on which the log path is located causes log files to be permanently lost, you can use the **mirrorlogpath** database configuration parameter to ensure that the logs are written to a secondary path. If the secondary path does not fail along with the primary disk or device, the log files are available for recovery.

When you set both the **mirrorlogpath** and **logarchmeth2** configuration parameters, the **logarchmeth2** configuration parameter archives log files from the mirror log path instead of archiving additional copies of the log files in the current log path. You can use this log archiving behavior to improve resilience during rollforward recovery. The reason is that a usable archived log file from the mirror log path might still be available to continue a database recovery operation, even if a primary log file from the current log path became corrupted because of a disk failure before archiving.

- The configured size of each log file has a direct bearing on log archiving. If each log file is very large, a large amount of data can be lost if a disk fails. If you configure your database to use small log files, the log manager archives the logs more frequently.

However, if you are moving the data to a slower device such as tape, you might want to have larger log files to prevent the queue from building up. Using larger

log files is also recommended if archiving each file requires substantial overhead, such as rewinding the tape device or establishing a connection to the archive media.

- If you use log archiving, the log manager attempts to archive primary logs as they are filled. In some cases, the log manager archives a log before it is full. This occurs if the log file is truncated because the database is deactivated, you issue the **ARCHIVE LOG** command, the end of an online backup is reached, or you issue the **SET WRITE SUSPEND** command.

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

- If you are archiving logs and backup images to a tape drive, you must ensure that the same tape drive is not the destination for both the backup images and the archived logs. Because some log archiving can take place while a backup operation is in progress, an error can occur when the two processes are trying to write to the same tape drive at the same time.

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

- The Db2 database manager opens a log file in read mode when it starts a user exit program to archive the file. On some operating systems, this prevents the user exit program from being able to delete the log file. Other operating systems, such as the AIX operating system, 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 might still be active and needed for crash recovery. The Db2 database manager manages disk space reuse when it archives the log files.
- A user exit or vendor program might receive 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. A user exit or vendor program might also receive a request to retrieve a file that does not exist, because it is located in another directory or the end of the logs was reached. In both cases, the user exit or vendor program should ignore this request and pass a successful return code.
- On Windows operating systems, you cannot use a REXX user exit to archive logs.
- The user exit or vendor program should allow for the existence of different log files with the same name after a point-in-time recovery. The user exit or vendor program should be written to preserve both log files and to associate those log files with the correct recovery path.
- If you enable a user exit or vendor program for two or more databases that are using the same tape device to archive log files and a rollforward operation is taking place on one of the databases, no other database should be active. If another database tries to archive a log file while the rollforward operation is in progress, one of the following situations might occur:
  - The logs that are required for the rollforward operation might not be found.
  - The new log file that is archived to the tape device might overwrite the log files that were previously stored on that tape device.

To prevent either situation from occurring, you can take one of the following steps:

- You can ensure that no other databases on the database partition that calls the user exit program are open during the rollforward operation.
- You can write a user exit program to handle this situation.

## Configuring a clustered environment for high availability

Creating a cluster of machines, and using cluster managing software to balance work load on those machines is one strategy for designing a highly available solution.

If you install IBM Db2 server on one or several of the machines in a cluster, you must configure the cluster manager to properly react to failures that affect the database or databases. Also, you must configure the database manager instances to work properly in the clustered environment.

### About this task

Configuring and administering the database instances and the cluster manager manually is complex, time consuming, and prone to error. The Db2 High Availability Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

**Note:** If you are using an AIX system, consider enabling the system error log (syslog) to capture relevant messages from Tivoli SA MP and RSCT subsystems and from the Db2 automation scripts. For more information, see the “Related links.”

### Procedure

1. Install cluster managing software.  
SA MP is integrated with Db2 Enterprise Server Edition, Db2 Advanced Enterprise Server Edition, Db2 Workgroup Server Edition, Db2 Connect Enterprise Edition, and Db2 Connect Application Server Edition on AIX, Linux, and Solaris SPARC operating systems. On Windows operating systems, SA MP is bundled with all of these Db2 database products and features, but it is not integrated with the Db2 installer.
2. Configure Db2 database manager instances for your cluster manager, and configure your cluster manager for Db2 server.  
Db2 high availability instance configuration utility (**db2haicu**) is a text-based utility that you can use to configure and administer your highly available databases in a clustered environment.
3. Over time, as your database needs change and you need to modify your database configuration within the clustered environment, continue to keep the database manager instance configuration and the cluster manager configuration synchronized.

## Synchronizing clocks in a partitioned database environment

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability.

Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max\_time\_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database environment, Db2 uses the system clock and the virtual timestamp stored in the SQLLOGCTL.LFH file on each machine as the basis for the

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

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 2005, and the log clock remains at November 7, 2005 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 2003.

Although Db2 cannot control updates to the system clock, the *max\_time\_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

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

## Client/server timestamp conversion

Timestamp conversion helps you maintain accurate records of activities on the database. It allows you to view activities in local time recorded in GMT time zone format, even if the database server is in a remote location with a different time zone.

Timestamps are essential for auditing purposes. It is important that the integrity of timestamps is maintained across all data partitions in a partitioned database environment.

This section explains the generation of timestamps in a client/server environment:

- If you specify a local time for a rollforward operation, all messages returned will also be in local time.

**Note:** All times are converted on the server and (in partitioned database environments) on the catalog database partition.

- The timestamp string is converted to GMT on the server, so the time represents the server's time zone, not the client's. If the client is in a different time zone from the server, the server's local time should be used.

- If the timestamp string is close to the time change due to daylight savings time, it is important to know whether the stop time is before or after the time change so that it is specified correctly.

---

## Administering and maintaining a highly available solution

Once you have created, configured, and started your Db2 database high availability solution running, there are ongoing activities you will have to perform. You need to monitor, maintain, and repair your database solution to keep it available to client applications.

### Procedure

As your database system runs, you need to monitor and respond to the following kinds of things:

1. Manage log files.  
Log files grow larger, require archiving; and some log files require copying or moving to be available for a restore operation.
2. Perform maintenance activities:
  - Installing software
  - Upgrading hardware
  - Reorganizing database tables
  - Database performance tuning
  - Database backup
3. Synchronize primary and secondary or standby databases so that failover works smoothly.
4. Identify and respond to unexpected failures in hardware or software.

### Log file management

The Db2 database manager uses a number scheme to name log files. This naming strategy has implications for log file reuse and log sequences. Also, a Db2 database that has no client application connection uses a new log file when the next client application connects to that database server.

These two aspects of Db2 database logging behavior affect the log file management choices you make.

Consider the following when managing database logs:

- The numbering scheme for archived logs starts with S00000000.LOG, and continues through S99999999.LOG, accommodating a potential maximum of 10 million log files. The database manager resets to S00000000.LOG if:
  - A database configuration file is changed to enable rollforward recovery
  - A database configuration file is changed to *disable* rollforward recovery
  - S99999999.LOG has been used.

The Db2 database manager reuses log file names after restoring a database (with or without rollforward recovery). The database manager ensures that an incorrect log is not applied during rollforward recovery. If the Db2 database manager reuses a log file name after a restore operation, the new log files are archived to separate directories so that multiple log files with the same name can be archived. The location of the log files is recorded in the recovery history file so that they can be applied during rollforward recovery. You must ensure that the correct logs are available for rollforward recovery.

When a rollforward operation completes successfully, the last log that was used is truncated, and logging begins with the next sequential log. Any log in the log path directory with a sequence number greater than the last log used for rollforward recovery is reused. Any entries in the truncated log following the truncation point are overwritten with zeros. Ensure that you make a copy of the logs before invoking the rollforward utility. (You can invoke a user exit program to copy the logs to another location.)

- If a database has not been activated (by way of the **ACTIVATE DATABASE** command), the Db2 database manager truncates the current log file when all applications have disconnected from the database. The next time an application connects to the database, the Db2 database manager starts logging to a new log file. If many small log files are being produced on your system, you might want to consider using the **ACTIVATE DATABASE** command. This not only saves the overhead of having to initialize the database when applications connect, it also saves the overhead of having to allocate a large log file, truncate it, and then allocate a new large log file.
- An archived log can be associated with two or more different *log sequences* for a database, because log file names are reused (see Figure 8 on page 165). For example, if you want to recover Backup 2, there are two possible log sequences that could be used. If, during full database recovery, you roll forward to a point in time and stop before reaching the end of the logs, you have created a new log sequence. The two log sequences cannot be combined. If you have an online backup image that spans the first log sequence, you must use this log sequence to complete rollforward recovery.

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

For example, suppose that a table space-level backup operation, Backup 3, completes between S0000013.LOG and S0000014.LOG in top log sequence (see Figure 8 on page 165). If you want to restore and roll forward using the database-level backup image, Backup 2, you need to roll forward through S0000012.LOG. After this, you could continue to roll forward through either the log sequence from which you took Backup 3 or the newer log sequence. If you roll forward through the newer log sequence, you cannot 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 have to restore the database-level backup image, Backup 2, and then roll forward using the top log sequence. After the table space-level backup image, Backup 3, has been restored, you can initiate a rollforward operation to the end of the logs.



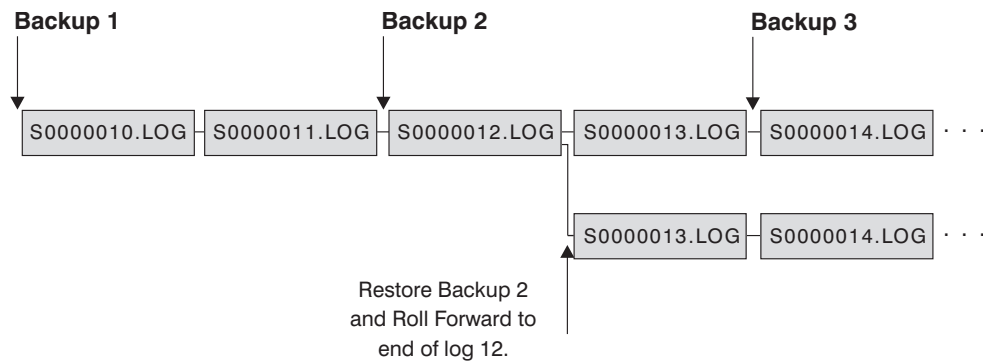


Figure 8. Reusing log file names

## On demand log archive

IBM Db2 server supports the closing (and, if enabled, the archiving) of the active log for a recoverable database at any time. This allows you to collect a complete set of log files up to a known point, and then to use these log files to update a standby database.

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

## Log archiving using db2tapemgr

You can use the **db2tapemgr** utility to store archived log files to tape devices. The **db2tapemgr** utility copies log files from disk to the specified tape device, and updates the recovery history file with the new location of the copied log files.

## Configuration

Set the database configuration parameter **logarchmeth1** to the location on disk of the log files you want to copy to tape. The **db2tapemgr** utility reads this **logarchmeth1** value to find the log files to copy. In a partitioned database environment, the **logarchmeth1** configuration parameter must be set on each database partition that contains log files to be copied.

The **db2tapemgr** utility does not use the **logarchmeth2** database configuration parameter.

## STORE and DOUBLE STORE parameters

Issue the **db2tapemgr** command with either the **STORE** or **DOUBLE STORE** parameter to transfer archived logs from disk to tape.

- The **STORE** parameter stores a range or all log files from the log archive directory to a specified tape device and deletes the files from disk.
- The **DOUBLE STORE** parameter scans the history file to see whether logs were stored to tape previously.
  - If a log has never been stored before, **db2tapemgr** stores the log file to tape and but does not delete it from disk.
  - If a log has been stored before, **db2tapemgr** stores the log file to tape and deletes it from disk.

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

When you issue the **db2tapemgr** command with either the **STORE** or **DOUBLE STORE** parameter, the **db2tapemgr** utility first scans the history file for entries where the **logarchmeth1** configuration parameter is set to disk. If it finds that any files that are supposed to be on disk, are not on disk, it issues a warning. If the **db2tapemgr** utility finds no log files to store, it stops the operation and issues a message to inform you that there is nothing to do.

## RETRIEVE parameters

Issue the **db2tapemgr** command with the **RETRIEVE** parameter to transfer files from tape to disk.

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

## Behavior

- If the **db2tapemgr** utility finds log files on disk, it then reads the tape header to make sure that it can write the log files to the tape. It also updates the history for those files that are currently on tape. If the update fails, the operation stops and an error message is displayed.
- If the tape is writeable, the **db2tapemgr** utility copies the logs to tape. After the files are copied, the log files are deleted from disk. Finally, the **db2tapemgr** utility copies the history file to tape and deletes it from disk.
- The **db2tapemgr** utility does not append log files to a tape. If a store operation does not fill the entire tape, then the unused space is wasted.
- The **db2tapemgr** utility stores log files only once to any given tape. This restriction exists to avoid any problems inherent to writing to tape media, such as stretching of the tape.
- In a partitioned database environment, the **db2tapemgr** utility only executes against one database partition at a time. You must run the appropriate command for each database partition, specifying the database partition number using the **ON DBPARTITIONNUM** parameter of the **db2tapemgr** command. You must also ensure that each database partition has access to a tape device.
- The **db2tapemgr** utility is not supported in Db2 pureScale environments.

## Examples

The following example shows how to use the **db2tapemgr** command to store all log files from the primary archive log path for database sample on database partition number 0 to a tape device and remove them from the archive log path:

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

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

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

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

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

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

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

## Automating log file archiving and retrieval with user exit programs

You can automate log file archiving and retrieval by creating a user exit program that the Db2 database manager calls to carry out the archiving or retrieval operation.

When the Db2 database manager invokes your user exit program, the following happens:

- The database manager passes control to the user exit program;
- The database manager passes parameters to the user exit program; and
- On completion, the user exit program passes a return code back to the database manager.

## Configuration

Before invoking a user exit program for log file archiving or retrieval, ensure that the **logarchmeth1** database configuration parameter is set to USEREXIT. This also enables your database for rollforward recovery.

## User exit program requirements

- The executable file for your user exit program must be called db2uext2.
- User exit programs must copy log files from the active log path to the archive log path, they must not move them. Do not remove log files from the active log path. If you remove log files from the active log path, your Db2 database might not be able to successfully recover in the event of a failure.  
Db2 database requires the log files to be in the active log path during recovery. The Db2 database server removes archived log files from the active log path when these log files are no longer needed for recovery.
- User exit programs must handle error conditions. Your user exit program must handle errors because the Db2 database manager can handle only a limited set of return conditions.  
See “User exit error handling” on page 169.
-

Each Db2 database manager instance can invoke only one user exit program. Because the database manager instance can invoke only one user exit program, you must design your user exit program with a section for each operation it might have to perform.

### **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 Data Server.

#### **• UNIX operating systems**

The user exit sample programs for Db2 Data Server for UNIX operating systems are found in the `sql1ib/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 operating systems:

- `db2uext2.ctsm`

This sample uses Tivoli Storage Manager to archive and retrieve database log files.

- `db2uext2.ctape`

This sample uses tape media to archive and retrieve database log files .

- `db2uext2.cdisk`

This sample uses the operating system COPY command and disk media to archive and retrieve database log files.

- `db2uext2.cxbsa`

This sample works with the XBSA Draft 0.8 published by the X/Open group. It can be used to archive and retrieve database log files. This sample is only supported on AIX.

#### **• Windows operating systems**

The user exit sample programs for Db2 Data Server for Windows operating systems are found in the `sql1ib\samples\c` subdirectory. Although the samples provided are coded in C, your user exit program can be written in a different programming language.

Your user exit program must be an executable file whose name is `db2uext2`.

There are two sample user exit programs for Windows operating systems:

- `db2uext2.ctsm`

This sample uses Tivoli Storage Manager to archive and retrieve database log files.

- `db2uext2.cdisk`

This sample uses the operating system COPY command and disk media to archive and retrieve database log files.

### User exit program calling format:

When the Db2 database manager calls a user exit program, it passes a set of parameters (of data type CHAR) to the program.

### Command syntax

```
db2uext2 -OS<os> -RL<db2rel> -RQ<request> -DB<dbname>  
-NN<nodenum> -LP<logpath> -LN<logname> -AP<tsmpasswd>  
-SP<startpage> -LS<logsize>
```

**os** Specifies the platform on which the instance is running. Valid values are: AIX, Solaris, HP-UX, SCO, Linux, and NT.

**db2rel** Specifies the Db2 release level. For example, SQL07020.

**request** Specifies a request type. Valid values are: ARCHIVE and RETRIEVE.

**dbname** Specifies a database name.

**nodenum** Specifies the local node number, such as 5, for example.

**logpath** Specifies the fully qualified path to the log files. The path must contain the trailing path separator. For example, /u/database/log/path/, or d:\logpath\.

**logname** Specifies the name of the log file that is to be archived or retrieved, such as S0000123.LOG, for example.

**tsmpasswd** Specifies the TSM password. (If a value for the database configuration parameter *tsm\_password* has previously been specified, that value is passed to the user exit program.)

**startpage** Specifies the number of 4-KB offset pages of the device at which the log extent starts.

**logsize** Specifies the size of the log extent, in 4-KB pages. This parameter is no longer used.

### User exit error handling:

If you create a user exit program to automate log file archiving and retrieval, your user exit program passes return codes to the Db2 database manager that invoked the user exit program.

The Db2 database manager can only handle a limited list of specific error codes. However, your user exit program might encounter many different kinds of error conditions, such as operating system errors. Your user exit program must map the error conditions it encounters to error codes that the database manager can handle.

Table 10 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 10. User Exit Program Return Codes*

Return Code	Explanation
0	Successful.
4	Temporary resource error encountered. <sup>a</sup>
8	Operator intervention is required. <sup>a</sup>
12	Hardware error. <sup>b</sup>
16	Error with the user exit program or a software function used by the program. <sup>b</sup>
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. <sup>b</sup>
24	The user exit program was not found. <sup>b</sup>
28	Error caused by an input/output (I/O) failure, or by the operating system. <sup>b</sup>
32	The user exit program was terminated by the user. <sup>b</sup>
255	Error caused by the user exit program not being able to load the library file for the executable. <sup>c</sup>

<sup>a</sup> For archiving or retrieval requests, a return code of 4 or 8 causes a retry in five minutes. If the user exit program continues to return 4 or 8 on retrieve requests for the same log file, Db2 will continue to retry until successful. (This applies to rollforward operations, or calls to the **db2ReadLog** API, which is used by the replication utility.)

<sup>b</sup> User exit requests are suspended for five minutes. During this time, all requests are ignored, including the request that caused the error condition. Following this five-minute suspension, the next request is processed. If this request is processed without error, processing of new user exit requests continues, and Db2 reissues the archive request that failed or was suspended previously. If a return code greater than 8 is generated during the retry, requests are suspended for an additional five minutes. The five-minute suspensions continue until the problem is corrected, or the database is stopped and restarted. Once all applications have disconnected from the database, Db2 issues an archive request for any log file that might not have been successfully archived previously. If the user exit program fails to archive log files, your disk might become filled with log files, and performance might be degraded. Once the disk becomes full, the database manager will not accept further application requests for database updates. If the user exit program was called to retrieve log files, rollforward recovery is suspended, but not stopped, unless the ROLLFORWARD STOP option was specified. If the STOP option was not specified, you can correct the problem and resume recovery.

<sup>c</sup> If the user exit program returns error code 255, it is likely that the program cannot load the library file for the executable. To verify this, manually invoke the user exit program. More information is displayed.

**Note:** During archiving and retrieval operations, an alert message is issued for all return codes except 0, and 4. The alert message contains the return code from the user exit program, and a copy of the input parameters that were provided to the user exit program.

## Log file allocation and removal

A log file which is required for crash recovery is called an active log. Unless infinite logging is enabled, log files in the active log path are never removed if they might be required for crash recovery.

If infinite logging is enabled and space needs to be made available for more active log files, the database manager archives an active log file and renames it to create a new active log file. If crash recovery is needed when infinite logging is used, log files might need to be retrieved from the archive log path to complete crash recovery. When you enable infinite logging, if the database manager is able to archive files successfully, then there is no limit to the number of active log files that can be created. However, if the database manager is not able to archive files successfully, then there is a limit of 258 unarchived active logs. After 258 unarchived logs are reached, the database manager will not be able to create any more new log files until the oldest log file is successfully archived.

When the **logarchmeth1** database configuration parameter is not set to OFF, a full log file becomes a candidate for removal only after it is no longer required for crash recovery, unless infinite logging is enabled, in which case the log files might be moved to the archive log path instead.

When **logarchmeth1** or **logarchmeth2** is set to a value other than OFF, LOGRETAIN, or USEREXIT, archived log file compression can be enabled to help reduce the amount of disk space required for archived log files.

The process of allocating new log files and removing old log files is dependent on the settings of the **logarchmeth1** and **logarchmeth2** database configuration parameters:

**logarchmeth1 and logarchmeth2 are set to OFF**

Circular logging is used. Roll-forward recovery is not supported with circular logging, while crash recovery is.

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

A log full situation can occur if all of the log files are active and the circular logging process cannot wrap to the first log file. Secondary log files are created when all the primary log files are active and full. Secondary log files are deleted when the database is deactivated or when the space they are using is required for the active log files.

**logarchmeth1 or logarchmeth2 is set to LOGRETAIN**

Archive logging is used. The database is a recoverable database. Both roll-forward recovery and crash recovery are enabled. The database manager does not manage the log files. After you archive the log files, you must delete them from the active log path so that the disk space can be reused for new log files. To determine which log files 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 the **loghead** value are not active and can be archived and removed.

**logarchmeth1 or logarchmeth2 is set to a value other than OFF or LOGRETAIN**

Archive logging is used. The database is a recoverable database. Both roll-forward recovery and crash recovery are enabled. When a log file becomes full, it is automatically archived by the database manager.

Log files are not deleted. Instead, when a new log file is required and one is not available, an archived log file is renamed and used again. An archived log file, is not deleted or renamed once it is closed and copied to the archived log file directory. The database manager renames the oldest

archived log when it is no longer needed for crash recovery. A log file that is moved to the database directory during recovery is removed during the recovery process when it is no longer needed.

If an error occurs when log files are being archived, archiving is suspended for the amount of time specified by the **archretrydelay** database configuration parameter. You can also use the **numarchretry** database configuration parameter to specify the number of times that the database manager is to try archiving a log file to the primary or secondary archive directory before it tries to archive log files to the failover directory (specified by the **failarchpath** database configuration parameter). **Numarchretry** is only used if the **failarchpath** database configuration parameter is set. If **numarchretry** is set to 0, the database manager continuously tries archiving from the primary or the secondary log path.

The easiest way to remove old log files is to restart the database. Once the database is restarted, only new log files and log files that the database manager failed to archive are found in the database directory.

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

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

### Using an overflow log path:

Configuring an overflow log path can help with the management of recovery log files when the retrieval of archived log files is required to facilitate a **ROLLFORWARD** recovery operation, or a db2ReadLog API request (used by replication products such as Q Replication or Change Data Capture), and other use cases. The basic behavior of the overflow log path is outlined in .

The overflow log path can be configured by using the **OVERFLOWLOGPATH** configuration parameter, or the OVERFLOW LOG PATH option of the **ROLLFORWARD** command.

If an overflow log path is configured, then log files that are retrieved from an archive log location are placed into the overflow log path. Otherwise, they are placed into the primary log path. This is referred to as the log retrieve location.

Consider some of the benefits and behaviors of using an overflow log path:



### **Using an overflow log path can help isolate and secure the primary (active) log path:**

By using an overflow log path, log files that are retrieved from the archive are placed into the overflow log path instead of the primary log path. This method can help to isolate the primary log path from intrusion or accidental manipulation of live active log data. It can also prevent the primary log path from unexpected disk/storage full conditions.

In high-performance environments, primary log paths are often configured with the fastest (first-tier) storage/disk. By using an overflow log path in these environments, a database administrator can use second-tier storage if desired.

### **Using an overflow log path can assist with the preparation of a ROLLFORWARD:**

When it is known beforehand that a **ROLLFORWARD** operation requires log files from an archive log path, a database administrator can manually retrieve these log files from the archive log path either before a **ROLLFORWARD** operation is started, or while it is running (if the **ROLLFORWARD** does not reach the replay log position contained in these log files). This method alleviates the need for **ROLLFORWARD** to retrieve log files from the archive log path in real time, and eliminates the possibility for failure during the **ROLLFORWARD** due to a transient retrieve log error or transient network issue.

### **Using an overflow log path can avoid the retrieval of archived log files:**

If the archive log paths (**LOGARCHMETH1**, **LOGARCHMETH2**, or both) are configured to use DISK method, then the overflow log path can be configured to point to the exact location of the archive log path. In this configuration, **ROLLFORWARD** avoids copying log files from the archive log path to the retrieve location. When the overflow log path is configured in this way, Db2 reads the log files directly from the overflow log path (which is the archive log path), and saves the cost of unnecessary I/O copy operations. When you configure the overflow log path in this way (to point to the archive log path), it is configured to point to the chain subdir subpath of the archive log path. For example, if the archive log path (**LOGARCHMETH1** or **LOGARCHMETH2**) is configured as `DISK:/some_logarch_path/`, then the overflow log path would be configured as `/some_logarch_path/<instance name>/<db name>/`.

**Note:** The benefit that is described is negated when native archive log file compression is enabled because the compressed log files must be decompressed and rewritten into the retrieve path. Thus, this technique is not to be used when native archive log file compression is enabled.

### **Including log files with a backup image**

When performing an online backup operation, you can specify that the log files required to restore and recover a database are included in the backup image.

This means that if you need to ship backup images to a disaster recovery site, you do not have to send the log files separately or package them together yourself. Further, you do not have to decide which log files are required to guarantee the consistency of an online backup. This provides some protection against the deletion of log files required for successful recovery.

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

To restore the log files from a backup image, use the **LOGTARGET** option of the **RESTORE DATABASE** command and specify a fully qualified path that exists on the Db2 server. The restore database utility then writes the log files from the image to the target path. If a log file with the same name exists in the target path, the restore operation fails and an error is returned. If the **LOGTARGET** option is not specified, no log files are restored from the backup image.

If the **LOGTARGET** option is specified and the backup image does not include any log files, an error is returned before an attempt is made to restore any table space data. The restore operation also fails if an invalid or read-only path is specified. During a database or table space restore where the **LOGTARGET** option is specified, if one or more log files cannot be extracted, the restore operation fails and an error is returned.

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

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

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

If you specify the **INCLUDE LOGS** option of the **BACKUP DATABASE** command when you back up a database, then perform a restore operation and a roll-forward operation that use that back up image, Db2 still searches for additional transaction logs when rolling the database forward, even though the backup image includes logs. It is standard rollforward behavior to continue to search for additional transaction logs until no more logs are found. It is possible to have more than 1 log file with the same timestamp. Consequently, Db2 does not stop as soon as it finds the first timestamp that matches the point-in-time to which you are rolling forward the database as there might be other log files that also have that timestamp. Instead, Db2 continues to look at the transaction log until it finds a timestamp greater than the point-in-time specified.

When no additional logs can be found, the rollforward operation ends successfully. However, if there is an error while searching for additional transaction log files, error SQL1268N is returned. Error SQL1268N can occur because during the initial restore, certain database configuration parameters were reset or overwritten. Three of these database configuration parameters are the TSM parameters, **tsm\_nodename**, **tsm\_owner**, and **tsm\_password**. They are all reset to NULL. To rollforward to the end of logs, you need to reset these database configuration parameters to correspond to the source system before the rollforward operation. Alternatively, you can specify the **NORETRIEVE** option when you issue the **ROLLFORWARD DATABASE** command. This prevents the Db2 database system from trying to obtain potentially missing transaction logs elsewhere.

**Note:**

1. This feature is not supported for offline backups.
2. When logs are included in an online backup image, the resulting image cannot be restored on releases of Db2 database before Version 8.2.

**Preventing the accidental loss of log files**

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

Consider the following scenarios:

- If you plan to drop a database before a restore operation, you need to save the log files in the active log path before issuing the **DROP DATABASE** command. After the database is restored, these log files might be required for rollforward recovery because some of them might not have been archived before the database was dropped. Normally, you are not required to drop a database before issuing the **RESTORE** command. However, you might have to drop the database (or drop the database on one database partition by specifying the **AT DBPARTITIONNUM** parameter of the **DROP DATABASE** command), because it was damaged to the extent that the **RESTORE** command fails. You might also decide to drop a database before the restore operation to give yourself a fresh start.
- If you are rolling a database forward to a specific point in time, log data after the time stamp you specify is overwritten. If, after you complete the point-in-time rollforward operation and reconnect to the database, you determine that you actually need to roll the database forward to a later point in time, you are not able to because the logs are already overwritten. It is possible that the original set of log files might have been archived; however, Db2 might be calling a user exit program to automatically archive the newly generated log files. Depending on how the user exit program is written, this might cause the original set of log files in the archive log directory to be overwritten. Even if both the original and new set of log files exist in the archive log directory (as different versions of the same files), you might have to determine which set of logs should be used for future recovery operations.

**Minimizing the impact of maintenance on availability**

You will have to perform maintenance on your Db2 database solution such as: software or hardware upgrades; database performance tuning; database backups; statistics collection; and monitoring for business purposes.

Minimizing the impact that performing that maintenance has on the availability of your solution involves careful scheduling of offline maintenance, and using Db2 features and functionality that reduce the availability impact of online maintenance.

**Before you begin**

Before you can use the following steps to minimize the impact of maintenance on the availability of your Db2 database solution, you must:

- configure automatic maintenance; and
- install the High Availability Disaster Recovery (HADR) feature.

## Procedure

1. Allow automatic maintenance to do your maintenance for you.

Db2 database can automate many database maintenance activities. Once the automatic maintenance has been configured, the maintenance will happen without you taking any additional steps to perform that maintenance.

2. Use a Db2 High Availability Disaster Recovery (HADR) rolling upgrade to minimize the impact of other maintenance activities.

If you are upgrading software or hardware, or if you are modifying some database manager configuration parameters, the HADR feature enables you to accomplish those changes with minimal interruption of availability. This seamless change enabled by HADR is called a rolling upgrade.

Some maintenance activities require you to shut down a database before performing the maintenance, even in the HADR environment. Under some conditions, the procedure for shutting down an HADR database is a little different than the procedure for shutting down a standard database: if an HADR database is started by a client application connecting to it, you must use the **DEACTIVATE DATABASE** command.

## Stopping Db2 High Availability Disaster Recovery (HADR)

If you are using the Db2 High Availability Disaster Recovery (HADR) feature, stopping HADR operations to perform maintenance on the primary or standby databases might be necessary. Stop HADR operations only on the database that you are performing maintenance. To stop using HADR completely, stop HADR on both databases.

### About this task

**Warning:** If you want to stop the specified database but you still want it to maintain its role as either an HADR primary or standby database, do not issue the STOP HADR command. If you issue the **STOP HADR** command the database will become a standard database and might require reinitialization in order to resume operations as an HADR database. Instead, issue the **DEACTIVATE DATABASE** command.

During upgrade to Db2 Version 11.1, the **STOP HADR** command can be issued on a Db2 Version 10.5 Fix Pack 7 or later database to change the database role to standard and to complete the upgrade process as a non-HADR database. To learn more, see Dealing with failures while upgrading Db2 servers in HADR environments (Version 10.5 Fix Pack 7 or later).

If for some reason the standby database is no longer required or there is an issue with the primary database the **STOP HADR** command can be issued on the standby database.

For an active standby database, an error message is returned. For a deactivated standby database, the HADR role is changed to STANDARD and the database is placed into rollforward pending. Issue the **ROLLFORWARD DATABASE** command with the STOP option to get the database out of rollforward pending state and then issue the **UPGRADE DATABASE** command to complete the upgrade process.

**Note:** If there is an issue with the standby database the **STOP HADR** command can be issued on the primary database. This changes the role to STANDARD and the **UPGRADE DATABASE** command can be issued to complete the upgrade process as a non-HADR database.

If you issue the **STOP HADR** command against a standard database, an error will be returned.

## Procedure

To stop HADR operations on the primary or standby database:

- From the CLP, issue the **STOP HADR** command on the database where you want to stop HADR operations.

In the following example, HADR operations are stopped on database SOCKS:

```
STOP HADR ON DATABASE SOCKS
```

If you issue this command against an inactive primary database, the database switches to a standard database and remains offline.

If you issue this command against an inactive standby database the database switches to a standard database, is placed in rollforward pending state, and remains offline.

If you issue this command on an active primary database, logs stop being shipped to the standby database and all HADR engine dispatchable units (EDUs) are shut down on the primary database. The database switches to a standard database and remains online. Transaction processing can continue. You can issue the **START HADR** command with the **AS PRIMARY** option to switch the role of the database back to primary database.

If you issue this command on an active standby database, an error message is returned, indicating that you must deactivate the standby database before attempting to convert it to a standard database.

- From an application, call the **db2HADRStop** application programming interface (API).
- From IBM Data Studio, open the task assistant for the **STOP HADR** command.

## Database activation and deactivation in a high availability disaster recovery (HADR) environment

If a standard database is started by a client connection, the database is shut down when the last client disconnects. If an HADR primary database is started by a client connection, it is equivalent to starting the database by using the **ACTIVATE DATABASE** command.

To shut down an HADR primary database that was started by a client connection, you need to explicitly issue the **DEACTIVATE DATABASE** command.

On a standard database in rollforward pending state, the **ACTIVATE DATABASE** and **DEACTIVATE DATABASE** commands are not applicable. You can only continue rollforward, stop rollforward, or use the **START HADR** command to start the database as an HADR standby database. Once a database is started as an HADR standby, you can use the **ACTIVATE DATABASE** and **DEACTIVATE DATABASE** commands to start and stop the database.

Activate a primary database using the following methods:

- client connection
- **ACTIVATE DATABASE** command
- Task assistant for the **ACTIVATE DATABASE** command in IBM Data Studio
- **START HADR** command with the **AS PRIMARY** option

Deactivate a primary database using the following methods:

- **DEACTIVATE DATABASE** command

**Note:** If you deactivate an HADR primary database that is in disconnected peer state using the **DEACTIVATE DATABASE** command or the `sql_deactivate_db` API, the database will be in an inconsistent state. The database will require crash recovery upon restart and no offline backups can be taken of this database until it is restarted.

- Task assistant for the **DEACTIVATE DATABASE** command in IBM Data Studio
- **db2stop** command with the **FORCE** parameter

Activate a standby database using the following methods:

- **ACTIVATE DATABASE** command
- Task assistant for the **ACTIVATE DATABASE** command in IBM Data Studio
- **START HADR** command with the **AS STANDBY** option

Deactivate a standby database using the following methods:

- **DEACTIVATE DATABASE** command
- Task assistant for the **DEACTIVATE DATABASE** command in IBM Data Studio
- **db2stop** command with the **FORCE** parameter

### Recommended order for shutting down an HADR pair

**Warning:** Although the **STOP HADR** command can be used to stop HADR on the primary or the standby, or both, it should be used with caution. If you want to stop the specified database but still want it to maintain its role as either an HADR primary or a standby database, do not issue the **STOP HADR** command. If you issue the **STOP HADR** command, the database becomes a standard database and might require reinitialization in order to resume operations as an HADR database. Instead, issue the **DEACTIVATE DATABASE** command.

If you only want to shut down the HADR operation, this is the recommended way of shutting down the HADR pair:

1. Deactivate the primary database
2. Stop Db2 on the primary database
3. Deactivate the standby database
4. Stop Db2 on the standby database

### Table space rebalance considerations in a Db2 High Availability Disaster Recovery (HADR) environment

You can use the **ALTER TABLESPACE REBALANCE** or **ALTER TABLESPACE USING STOGROUP** statement to start a rebalance operation on a primary database. The statement is replayed on the standby database, and a corresponding rebalance operation is started.

During the rebalance operation, you can specify the **ALTER TABLESPACE** statement with the **REBALANCE SUSPEND** clause to suspend the rebalance operation on the primary database. To resume the suspended rebalance operation, specify the **ALTER TABLESPACE** statement with the **REBALANCE RESUME** clause.

The standby database remains in active state when it replays an **ALTER TABLESPACE REBALANCE SUSPEND** statement. Because the rebalance is suspended on the primary database, when the standby takes over as the new

primary database the rebalance operation on the new primary database is suspended and the rebalance operation on the new standby database is implicitly resumed.

When you restore a database using a split mirror as a clone database or as a standby database, any suspended rebalance operations for table spaces are automatically resumed at database startup.

### **Performing rolling updates in a Db2 high availability disaster recovery (HADR) environment**

Use this procedure in a high availability disaster recovery (HADR) environment when you upgrade software or hardware, update your Db2 database product software, or change database configuration parameters.

This procedure keeps database service available throughout the rolling update process, with only a momentary service interruption when processing is switched from one database to the other. With multiple standbys, you can provide continued HA and DR protection throughout the rolling update process.

#### **Before you begin**

**Note:** This procedure is distinct from the HADR rolling update procedure for Db2 pureScale environments, which is described in the following task: Installing online fix pack updates to a higher code level in a HADR environment. Note that the following update procedures are also not for an automated HADR environment. If you would like to proceed on performing rolling updates on an automated HADR environment, see “Performing rolling updates in an automated Db2 high availability disaster recovery (HADR) environment” on page 182.

Review the system requirements for HADR. See “System requirements for Db2 high availability disaster recovery (HADR)” on page 139.

If the **hadr\_syncmode** database configuration parameter is set to SYNC, NEARSYNC or ASYNC, the HADR pair must be in a peer state before you start the rolling update. If the **hadr\_syncmode** database configuration parameter is set to SUPERASYNC, ensure that the standby database is not too far behind the primary database before you start the rolling update.

If you have two HADR databases (databaseA and database B) set up the following way, perform a role switch on one database so that both primaries are on the same system during the fix pack update:

- The primary for databaseA runs on system1, and the standby runs on system2
- The primary for databaseB runs on system2, and the standby runs on system1

The overall capacity of the databases might be reduced, but it keeps both database online during the procedure.

**Note:** All Db2 fix pack updates, hardware upgrades, and software upgrades should be implemented in a test environment before being applied to your production system.

#### **About this task**

Use this procedure to perform a rolling update on your Db2 database system, to perform maintenance on your Db2 pureScale cluster, and to update the Db2 database product software from one modification level to another. For example,

applying a fix pack to a Db2 database product software. During rolling updates, the modification level or fix pack level of the standby database can be later than that of the primary database while testing the new level. However, you should not keep this configuration for an extended period to reduce the risk of using features that might be incompatible between the levels. The primary and standby databases will not connect to each other if the modification level of the Db2 database product software for the primary database is later than that of the standby database.

A rolling update cannot be used to upgrade from an earlier version to a later version of a Db2 database product software. For example, you cannot use this procedure to upgrade a Db2 database product from Version 10.5 to Db2 Version 11.1. To upgrade a Db2 server in HADR environment see, [com.ibm.db2.luw.qb.upgrade.doc/doc/c0070028.dita](http://com.ibm.db2.luw.qb.upgrade.doc/doc/c0070028.dita).

## Procedure

To perform a rolling update in an HADR environment:

1. Update the standby database by issuing the following steps:
  - a. Use the **DEACTIVATE DATABASE** command to shut down the standby database.
  - b. If necessary, shut down the instance on the standby database.
  - c. Change one or more of the following: the software, the hardware, or the Db2 configuration parameters.

**Note:** You cannot change any HADR configuration parameters when performing a rolling update.

- d. If necessary, restart the instance on the standby database.
  - e. Use the **ACTIVATE DATABASE** command to restart the standby database.
  - f. Ensure that HADR enters peer state. Use the `MON_GET_HADR` table function (on the primary or a read-enabled standby) or the **db2pd** command with the `-hadr` option to check this.
2. Switch the roles of the primary and standby databases:
  - a. Issue the **TAKEOVER HADR** command on the standby database.
  - b. Direct clients to the new primary database. This can be done using automatic client reroute.

**Note:** Because the standby database takes over as the primary database, the new primary database is now updated. If you are applying a Db2 fix pack, the **TAKEOVER HADR** command changes the role of the original primary database to standby database. However, the command does not let the new standby database connect to the newly updated primary database. Because the new standby database uses an older version of the Db2 database system, it might not understand the new log records generated by the updated primary database, and it will be shut down. In order for the new standby database to reconnect with the new primary database (that is, for the HADR pair to reform), the new standby database must also be updated.

3. Update the original primary database (which is now the standby database) using the same procedure as in step 1. When you have done this, both databases are updated and connected to each other in HADR peer state. The HADR system provides full database service and full high availability protection.
  4. Optional: To enable the HADR reads on standby feature during the rolling update perform the following steps to ensure the consistency of the internal Db2 packages on the standby database before read operations are introduced.



The binding of internal Db2 packages occurs at first connection time, and can complete successfully only on the primary database.

- a. Enable the HADR reads on standby feature on the standby database as follows:
    - 1) Set the **DB2\_HADR\_ROS** registry variable to ON on the standby database.
    - 2) Use the **DEACTIVATE DATABASE** command to shut down the standby database.
    - 3) Restart the instance on the standby database.
    - 4) Use the **ACTIVATE DATABASE** command to restart the standby database.
    - 5) Ensure that HADR enters peer state. Use the MON\_GET\_HADR table function (on the primary or a read-enabled standby) or the **db2pd** command with the **-hadr** option to check this.
  - b. Switch the roles of the primary and standby database as follows:
    - 1) Issue the **TAKEOVER HADR** command on the standby database.
    - 2) Direct clients to the new primary database.
  - c. Repeat the same procedure in substep a to enable the HADR reads on standby feature on the new standby database.
5. Optional: If did not perform step 4 on page 180 and you want to return to your original configuration, switch the roles of the primary and standby database as you did in step 2 on page 180.
6. Optional: In an HADR environment, run **db2updv111** only on the primary database. After running the **db2updv111** command, you might have to restart the database for changes from **db2updv111** command to take effect. To perform a restart:

**Attention:** **db2updv111** might deactivate packages and a **REBIND** must be run. After the **REBIND** is complete, all packages are valid and the instances do not need to be recycled.

- a. Restart the standby database by deactivating and reactivating it. The standby database is restarted to prevent the disruption of primary database service.
  - 1) Run the following command on the standby database:

```
DEACTIVATE
db dbname
```

where *dbname* is the name of the standby database.
  - 2) Run the following command on the standby database:

```
ACTIVATE
db dbname
```

where *dbname* is the name of the standby database.
- b. Switch the roles of the primary and standby databases:
  - 1) Run the following command on the standby database:

```
TAKEOVER
hadr on db dbname
```

where *dbname* is the name of the standby database.
  - 2) Direct clients to the new primary database.

**Note:** The databases have switched roles. The primary database was previously the standby database and the standby database was previously the primary database.

- c. Restart the standby database (formerly the primary database), using the same method as in Step 1.
- d. Switch the roles of the primary and standby databases to return the database to their original roles. Switch the roles using the same method as in step 2.

## Performing rolling updates in an automated Db2 high availability disaster recovery (HADR) environment

When you use the integrated High Availability (HA) feature to automate HADR, extra steps are required to update operating system or Db2 database system software, upgrade hardware, or change database configuration parameters. Use this procedure to perform a rolling upgrade in an automated HADR environment.

### Before you begin

**Note:** The following update procedures are for an automated HADR environment. If you want to perform rolling updates on an HADR environment that is not automated, see “Performing rolling updates in a Db2 high availability disaster recovery (HADR) environment” on page 179.

You must have the following prerequisites ready to perform the steps that are described in the procedures section:

- Two Db2 instances.
- Two Db2 servers.
- The instances are originally running at Version 11.1.1.1 or a later version. If the instances are running on Version 11.1 GA, refer to this IBM technote.
- The instances are configured with IBM Tivoli System Automation for Multiplatforms (SA MP) controlling HADR failover.

**Note:** All Db2 fix pack updates, hardware upgrades, and software upgrades must be implemented in a test environment prior to applying them to your production system.

The HADR pair must be in PEER state prior to starting the rolling update.

### Restrictions

Use this procedure to perform a rolling update on your Db2 database system and update the Db2 database product software to a new fix pack level in an automated HADR environment. For example, applying a fix pack to a Db2 database product software.

- The Db2 instances must be currently running at Version 11.1.1.1 or a later version. If the instances are running on Version 11.1 GA, refer to this IBM technote.

A rolling update cannot be used to upgrade a Db2 database system from an earlier version to a later version. For example, you cannot use this procedure to upgrade from Db2 Version 10.5 to Db2 Version 11.1. To upgrade a Db2 server in an automated HADR environment, see [Upgrading Db2 servers in an automated HADR environment](#).

You cannot use this procedure to update the Db2 HADR configuration parameters. Updates to the HADR configuration parameters must be made separately. Because HADR requires the parameters on the primary and standby to be the same, both

the primary and standby databases might need to be deactivated and updated at the same time.

## Procedure

1. On the standby node, stop all Db2 processes:
  - deactivate db <database-name>. This command stops HADR, but retains the role.
  - db2stop force.
2. Run the stoprpnnode -f <standby node> command as root.
3. Apply Fix Pack.
4. On the primary node, run the starttrpnnode <standby node> command as root.
5. On the standby node, start all Db2 processes:
  - db2start.
  - activate db <database-name>. This command resumes HADR but retains the role.
  - Verify that the HADR pair has established PEER state via the db2pd -hadr db <database-name> command.
6. Perform a role-switch:
  - On the standby node, issue the db2 takeover hadr on db <database-name> command.
  - Old primary disconnects because new primary is on a higher fix pack level.
7. On the old primary node, repeat steps 1-5 to apply the fix pack.
8. Perform a failback to locate the HADR roles back to their original state.
  - On the standby (old primary) node issue the db2 takeover hadr on db <database-name> command.
  - Prior to starting the fix pack installation process, verify that the original primary node is the PRIMARY and verify that the HADR pair is still in PEER state via the db2pd -hadr db <database-name> command.
9. If required, migrate the TSA domain.
  - TSA domain migration is only required if the new Db2 fix pack includes a new TSA version. It is not always the case that the new Db2 fix pack includes a new TSA version.
  - TSA domain migration is required if the active version number (AVN) does not match the installed version number (IVN). These values can be listed by running the `lssrc -ls IBM.RecoveryRM |grep VN` command.
  - To migrate TSA domain, issue the following command as root:

```
export CT_MANAGEMENT_SCOPE=2
runact -c IBM.PeerDomain CompleteMigration Options=0
samctrl -m # Type 'Y' to confirm migration
```
  - Verify that the AVN and IVN values match via the `lssrc ls IBM.RecoveryRM |grep VN` command.
10. Verify that MixedVersions is set to No for the cluster manager by running the `lsrpdomain` command.

## Scenario: Changing the system clock

When adjusting or changing the system clock, there is no reason to stop the Db2 database manager. Db2 successfully handles daylight saving time changes twice a year all over the world without issue.

Configurations which use NTP to synchronize clocks across systems are also fully supported.

### About this task

There are some best practices that you must be aware of when changing the system time.

#### Restrictions

When changing the system clock in the vast majority of scenarios there is absolutely no impact.

When major time shifts occur, you must be aware of two situations.

- If you execute point-in-time recovery you need to be aware of any significant time shifts.
- Function definitions include the time and date they were created in the form of a timestamp. At function invocation, Db2 attempts to resolve the function definition. As part of the function resolution, the timestamp value logged in the function definition at create time is checked. If you move the system clock back to a time before the functions were created, Db2 does not resolve references to those functions.

### Procedure

Best practices to avoid these two situations:

1. If you are moving time forward, proceed to step 3.
2. If you are moving time backward by *X* minutes:
  - a. Choose a time to execute the change when no new functions were created in the past *X* minutes, and no update transactions occur in *X* minutes.
  - b. If you are unable to find a time as outlined in step a, you can still move the system clock backwards by *X* minutes with Db2 online. However, you must accept the following implications:
    - You might not be able to use point-in-time recovery to recover to a point within those *X* minutes. That is, you might not be able to recover a subset of the update transactions that executed within those *X* minutes.
    - Functions created within *X* minutes before the change might not be resolved for *X* minutes after the change.
3. Change the system clock.

### Results

By following the best practices as outlined, you avoid any potential point-in-time recovery or function resolution issue when changing the system clock.

## Synchronizing the primary and standby databases

One high availability strategy is to have a primary database and a secondary or standby database to take over operations if the primary database fails.

If the standby database must take over database operations for a failed primary database, it must contain exactly the same data, know about all inflight transactions, and otherwise continue database processing exactly the same way as

the primary database server would, if it had not failed. The ongoing process of updating the standby database so that it is a copy of the primary database is called synchronization.

## Before you begin

Before you can synchronize the primary and standby databases you must:

- Create and configure the primary and standby databases.
- Configure communications between the primary and standby databases.
- 

Choose a synchronization strategy (for example, log shipping, log mirroring, suspended I/O and disk mirroring, or HADR.)

There are several strategies for keeping the primary database server and the standby database server synchronized:

- shipping logs from the primary database to the standby database and rolling them forward on the standby database;
- writing database logs to both the primary and standby databases at the same time, known as log mirroring;
- using suspended I/O support with disk mirroring to periodically taking a copy of the primary database, splitting the mirror and initializing the copy as a new standby database server; and
- using a availability feature such as the Db2 High Availability Disaster Recovery (HADR) feature to keep the primary and standby database synchronized.

## Procedure

1. If you are using logs to synchronize the primary database and the secondary or standby database, configure Db2 database to perform the required log management for you. For example, if you want Db2 database to mirror the logs, set the **mirrorlogpath** configuration parameter to the location where you want the second copy of the logs to be saved.
2. If you are using Db2 database suspended I/O functionality to split a disk mirror of the primary database, you must do the following:
  - a. Initialize the disk mirroring for the primary database.
  - b. When you need to split the mirror of the primary database, follow the instructions in the topic “Using a split mirror as a standby database.”
3. If you are using the HADR feature to manage synchronizing the primary and standby databases, configure Db2 database for HADR, and allow Db2 database to synchronize the primary and standby databases for you.

## Resolving log replay error when creating table space

If you create a table space on the primary database and log replay fails on the standby database because the containers are not available, the primary database does not receive an error message stating that the log replay failed.

If a takeover operation occurs, the new table space that you created is not available on the new primary database. To recover from this situation, restore the table space on the new primary database from a backup image.

To check for log replay errors, you must monitor the `db2diag.log` file and the administration notification log file on the standby database when you are creating new table spaces.

In the following example, table space MY\_TABLESPACE is restored on database MY\_DATABASE before it is used as the new primary database:

1. Connect to the database MY\_DATABASE:  
`DB2 CONNECT TO my_database`
2. Obtain the list of unavailable table spaces that need to be restored:  
`DB2 LIST TABLESPACES SHOW DETAIL`

You specify the relevant tablespace ID in step 5 of this procedure.

3. Stop HADR on the primary database:  
`DB2 STOP HADR ON DATABASE my_database`
4. Perform an online redirected restore of the tablespace:  
`DB2 RESTORE my_database TABLESPACE (my_tablespace) ONLINE REDIRECT`
5. Define the new table space containers:  
`DB2 SET TABLESPACE CONTAINERS FOR my_tablespace_ID IGNORE ROLLFORWARD CONTAINER OPERATIONS USING /new_container_path/')`
6. Complete the restore operation:  
`DB2 RESTORE my_database CONTINUE`
7. Roll forward the database to the end of logs:  
`DB2 ROLLFORWARD DATABASE my_database TO END OF LOGS AND STOP TABLESPACE (my_tablespace)`
8. Start HADR on the primary database:  
`DB2 START HADR ON DATABASE my_database AS PRIMARY`

## Db2 High Availability Disaster Recovery (HADR) replicated operations

Db2 High Availability Disaster Recovery (HADR) uses database logs to replicate data from the primary database to the standby database. Some activities can cause the standby database to fall behind the primary database as logs are replayed on the standby database.

Some activities are so heavily logged that the large amount of log files they generate can cause storage problems. Although replicating data to the standby database using logs is the core of availability strategies, logging itself can potentially have a negative impact on the availability of your solution. Design your maintenance strategy wisely, configure your system to minimize the negative impact of logging, and allow logging to protect your transaction data.

In high availability disaster recovery (HADR), the following operations are replicated from the primary to the standby database:

- Data definition language (DDL)
- Data manipulation language (DML)
- Buffer pool operations
- Table space operations
- Online reorganization
- Offline reorganization
- Metadata for stored procedures and user defined functions (UDF) (but not the related object or library files)

During an online reorganization, all operations are logged in detail. As a result, HADR can replicate the operation without the standby database falling further

behind than it would for more typical database updates. However, this behavior can potentially have a large impact on the system because of the large number of log records generated.

While offline reorganizations are not logged as extensively as online reorganizations, operations are typically logged per hundreds or thousands of affected rows. This means that the standby database could fall behind because it waits for each log record and then replays many updates at once. If the offline reorganization is non-clustered, a single log record is generated after the entire reorganization operation. This mode has the greatest impact on the ability of the standby database to keep up with the primary database. The standby database will perform the entire reorganization after it receives the log record from the primary database.

HADR does not replicate stored procedure and UDF object and library files. You must create the files on identical paths on both the primary and standby databases. If the standby database cannot find the referenced object or library file, the stored procedure or UDF invocation will fail on the standby database.

### **Db2 High Availability Disaster Recovery (HADR) non-replicated operations**

Db2 High Availability Disaster Recovery (HADR) uses database logs to replicate data from the primary database to the standby database. Non-logged operations are allowed on the primary database, but not replicated to the standby database.

If you want non-logged operations, such as updates to the history file, to be reflected in the standby database, you must take extra steps to cause this to happen.

The following are examples of cases in which operations on the primary database are not replicated to the standby database:

- Tables created with the NOT LOGGED INITIALLY option specified are not replicated. Attempts to access such tables after an HADR standby database takes over as the primary database result in an error.
- All logged LOB columns are replicated. Non-logged LOB columns are not replicated. However, the space for them is allocated on the standby database using binary zeros as the value for the column.
- Updates to database configuration using the **UPDATE DATABASE CONFIGURATION** and **UPDATE DATABASE MANAGER CONFIGURATION** commands are not replicated.
- Database configuration and database manager configuration parameters are not replicated.
- For user-defined functions (UDFs), changes to objects external to the database (such as related objects and library files) are not replicated. They need to be set up on the standby via other means.
- The recovery history file (db2rhist.asc), and changes to it, are not automatically shipped from the primary database to the standby database.

You can place an initial copy of the history file (obtained from the backup image of the primary) on the standby database by issuing the **RESTORE DATABASE** command with the **REPLACE HISTORY FILE** parameter:

```
RESTORE DB KELLY REPLACE HISTORY FILE
```

After HADR is initialized and subsequent backup activities take place on the primary database, the history file on the standby database becomes out of date. However, a copy of the history file is stored in each backup image. Alternatively,

you can create a no table space backup image that contains a more recent copy of the history file by using the following command:

```
BACKUP DB KELLY NO TABLESPACE
```

You can update the history file on the standby by extracting the history file from a backup image by using the following command:

```
RESTORE DB KELLY HISTORY FILE
```

Do not use regular operating system commands to copy the history file in the database directory from the primary database to the standby database. The history file can become corrupted if the primary is updating the files when the copy is made.

If a takeover operation occurs and the standby database has an up-to-date history file, backup and restore operations on the new primary generate new records in the history file and blend seamlessly with the records generated on the original primary. If the history file is out of date or has missing entries, an automatic incremental restore might not be possible; instead, a manual incremental restore operation is required.

### **Db2 high availability disaster recovery (HADR) database states**

At any time, a high availability disaster recovery (HADR) standby database is in one of five states: local catchup, remote catchup pending, remote catchup, peer, or disconnected peer. The states are defined by the log shipping status. Regardless of the state, log replay of all available logs occurs.

If a standby is connected to the primary, its is reported in the HADR\_STATE field of the MON\_GET\_HADR table function and the **db2pd** command output. (If it is not connected, it reports DISCONNECTED.)

Figure 9 on page 189 shows the progression through the different standby database states.



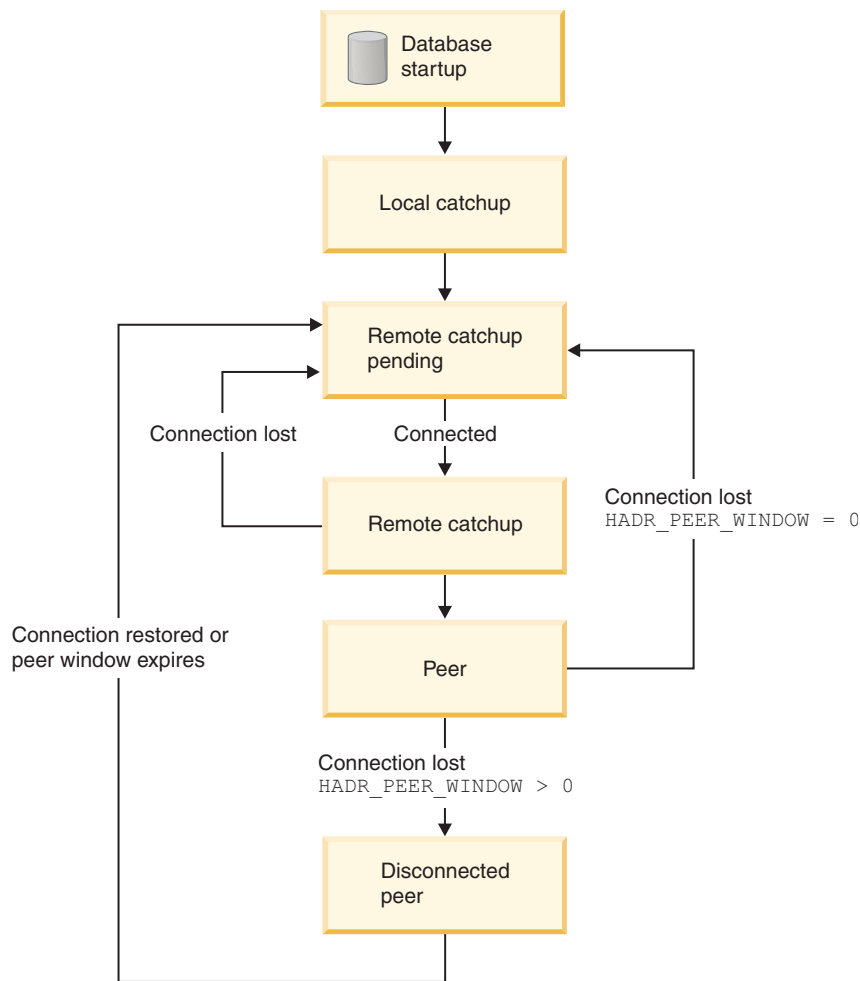


Figure 9. States of the standby database

## Local catchup state

With the HADR feature, when a database is started as a standby, it enters local catchup state, and the log files in its local log path are read to determine what logs are available locally. In this state, logs are not retrieved from the archive even if you configured a log archiving method. Also, in this state, a connection to the primary database is not required; however, if a connection does not exist, the standby database tries to connect to the primary database. When the end of local log files is reached, the standby database enters remote catchup pending state.

## Remote catchup pending state

When the standby enters remote catchup pending state, if a connection to the primary has not been established, the standby waits for a connection. After a connection is established, the standby obtains the primary's current log chain information. This enables the standby, if you configured a log archive, to retrieve log files from the archive and verify that the log files are valid.

In remote catchup state and peer state, if the standby loses its connection to the primary, it goes back to remote catchup pending state. When the connection is reestablished, the standby tries to retrieve the logs from the archive. Thus, if you configure a shared archive device, the standby might be able to find more logs

than would be available if it is using a separate archiving device. As a result, using an archive can have less impact on the primary than shipping from the primary through the HADR connection.

## Remote catchup state

In remote catchup state, the primary database reads log data from its log path or by way of a log archiving method, and the log data is sent to the standby database. The primary and standby databases enter peer state when the standby database receives all the on-disk log data of the primary database. If you are using the SUPERASYNC synchronization mode, the primary and standby never enter peer state. They permanently stay in remote catchup state, which prevents the possibility of blocking primary log writing in peer state.

If the connection between the primary and standby databases is lost when the databases are in remote catchup state, the standby database enters remote catchup pending state.

## Assisted remote catchup

Assisted remote catchup state is specific to HADR in Db2 pureScale environments.

A standby replay member might not be able to directly connect to a member on the primary because of network problems or the member on the primary being inactive. In this case, the standby replay member gets the unreachable member's logs through the assistance of another member on the primary that can connect to the standby. This *assisting member* uses a dedicated TCP connection for each member that it is assisting. Log streams that are in assisted remote catchup state can never enter peer state because indirect connections are used for them. Assisted remote catchup is automatically terminated when the standby replay member can directly connect to the member on the primary.

You can determine whether a member's log stream is in assisted remote catchup state by using the `MON_GET_HADR` table function or the `db2pd` command. For a member on the primary, its log stream is shown as being in `REMOTE_CATCHUP` state, and the `HADR_FLAGS` field contains the `ASSISTED_REMOTE_CATCHUP` flag.

## Peer state

In peer state, log data is shipped directly from the primary's log write buffer to the standby whenever the primary flushes its log pages to disk. The HADR synchronization mode specifies whether the primary waits for the standby to send an acknowledgement message that log data was received. The log pages are always written to the local log files on the standby database. This behavior guards against a crash and allows a file to be archived on the new primary in case of takeover, if it was not archived on the old primary. After being written to the local disk, the received log pages can then be replayed on the standby database. If log spooling is disabled, which is the default, log replay reads logs only from the log receive buffer.

If log replay is slow, the receive buffer can fill up, and the standby stops receiving new logs. If this happens, primary log writing is blocked. If you enable log spooling, a part of the log buffer is released even if it was not replayed yet, so primary log writing can continue. Log replay reads the log data from disk later. If

the spooling device fills up or the configured spool limit is reached, the standby still stops receiving, and the primary is blocked again.

If the connection between the primary and standby databases is lost when the databases are in peer state and the **hadr\_peer\_window** database configuration parameter is set to 0, which is the default, the standby database enters remote catchup pending state. However, if the connection between the primary and standby databases is lost during peer state and you set the **hadr\_peer\_window** parameter to a nonzero value (meaning that you configured a *peer window*), the standby database enters disconnected peer state.

### Disconnected peer state

If you configured a peer window and the primary database loses its connection with the standby database while in peer state, the primary database continues to behave as though the primary and standby databases were in peer state. This behavior lasts until the peer window expires or until the standby reconnects, whichever occurs first. When the primary database and standby database are disconnected but behave as though in they were in peer state, this state is called *disconnected peer*.

The advantage of configuring a peer window is that it lowers the risk of transaction loss during multiple or cascading failures. Without the peer window, when the primary database loses its connection with the standby database, the primary database moves out of peer state immediately and continues transaction processing. These transactions are not replicated to the standby. If the primary server fails shortly after it loses its connection to the standby, the risk of transaction loss is high in a failover. With the peer window enabled, the primary database blocks transaction processing for a certain amount of time after losing its connection to the standby in peer state, guarding against cascading failures. Furthermore, the standby can take over within the peer window time with no risk of data loss.

The disadvantage of configuring a peer window is that transactions on the primary database take longer or even time out while the primary database is in the peer window waiting for the connection with the standby database to be restored or for the peer window to expire. As well, intermittent network failures can cause a severe impact on primary transaction processing.

You can determine the peer window size, which is the value of the **hadr\_peer\_window** database configuration parameter, by using the **MON\_GET\_HADR** table function or the **db2pd** command with the **-hadr** parameter.

### Manually copying log files from the primary database to the standby database

One way to synchronize the primary and standby databases is to manually copy the primary database log files into the standby database log path or overflow log path, if configured. Manually copying files can be especially helpful if there is a large log gap between the primary and standby, for example, because the standby database was down for a long time. Manually copying files can reduce the delay of the standby having to retrieve the logs from the archive, or it can reduce the impact on the primary of having to ship these log files, which the primary would likely have to retrieve from the archive.

It is important to do this step before activating the standby database. After you deactivate the standby database, it proceeds with searching local log files, attempting to retrieve from the archive, and engaging the primary for log shipping, as described previously. If you copy the log files to the standby after you have activated it can interfere with the standby's normal operation.

## Determining the HADR standby database state

The state of a Db2 high availability disaster recovery (HADR) standby database determines what operations it can perform.

### Procedure

To determine the state of an HADR standby database in a primary-standby HADR database pair:

- From the primary database or a standby database, issue the **db2pd** command with the **-hadr** parameter and check the HADR\_STATE field:
  - If you issue the command from the primary database, the command returns a set of data for each standby in your HADR setup.
  - If you issue the command from a standby database, the command returns only a single set of data because the standby cannot obtain information about other standbys.
- Issue a query that uses the MON\_GET\_HADR table function to determine the HADR\_STATE field on the primary database or a read-enabled standby database:
  - If you issue the query on the primary database, the table function returns a row of data for each standby in your HADR setup.
  - If you issue the query on the standby database, the table function returns only a single row of data because a standby cannot obtain information about other standbys.

In a Db2 pureScale environment, you can use this table function on the primary only. You can specify any single member, the current member, or all members. The returned rows represent log streams that are being processed by the member.

### Example

#### Example with one HADR standby

A DBA with an HADR setup with a single standby issues the MON\_GET\_HADR table function from the primary to query the state of the HADR databases:

```
select HADR_STATE from table (mon_get_hadr(NULL))
```

The following information is returned, showing that the HADR pair is in peer state:

```
HADR_STATE
-----
PEER
```

1 record(s) selected.

#### Example with one HADR standby in a Db2 pureScale environment

A DBA with an HADR setup with three-member clusters (members 0, 1, and 2) issues the MON\_GET\_HADR table function from the primary to query the state of the HADR databases on all members:

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE, HADR_FLAGS
from table (mon_get_hadr(-2))
```

The following information is returned:

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE	HADR_FLAGS
0	0	0	PEER	
1	0	0	REMOTE_CATCHUP	ASSISTED_REMOTE_CATCHUP
2	2	0	PEER	

3 record(s) selected.

This output indicates that member 1 is in assisted remote catchup state and that member 0 is the assisting member. If the DBA issues the table function with a member argument of member 1, the result is as follows:

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE, HADR_FLAGS
from table (mon_get_hadr(1))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE	HADR_FLAGS
1	1	0	DISCONNECTED	

### Example with multiple HADR standbys

A DBA with an HADR setup with multiple standbys issues the MON\_GET\_HADR table function from the primary to query the state of the HADR databases:

```
select STANDBY_ID, HADR_STATE from table (mon_get_hadr(NULL))
```

The following information is returned:

STANDBY_ID	HADR_STATE
1	PEER
2	REMOTE_CATCHUP
3	REMOTE_CATCHUP

3 record(s) selected.

The principal standby, which always has a STANDBY\_ID value of 1, is in peer state. The auxiliary standbys, which have STANDBY\_ID values of 2 and 3, can never be in peer state.

### Recovering from table space errors on an HADR standby

In the event that the HADR standby database encounters an error on a particular table space during log replay, the standby database will continue to replay logs on other table spaces but will stop replaying logs on the affected table space.

#### About this task

The affected table space's tablespace state will be changed to restore pending, rollforward pending, or offline. You need to recover the table space on the standby because data in this table space will not be available if this database takes over the primary role.

#### Procedure

1. Correct the root cause of the error. Possible causes include:
  - Insufficient space
  - The file system is not mounted
  - A load copy could not be found

2. Repair the affected table space. Do this by completely reinitializing the standby database by restoring a backup image of the primary database.

### HADR role switch and quiesced table spaces

In an high availability disaster recovery (HADR) environment, a table space quiesce is not preserved during a role switch.

When a table space is quiesced on the primary database, no log records are generated, so there is no effect on the standby database. If the standby has to take over as the primary before the quiesce has been released, that table space will be fully available on the new primary. You should be aware that if you continue the job that required the table space to be quiesced on the original primary, then on the new primary, the job is no longer protected by the quiesce

If there was a role switch (that is, if the old primary is now the new standby), changes to the table space on the new primary are replayed on the new standby. However, if the primary role is failed back to the old primary, the quiesce state will still be in effect for that table space.

### HADR delayed replay

HADR delayed replay helps prevent data loss due to errant transactions. To implement HADR delayed replay, set the **hadr\_replay\_delay** database configuration parameter on the HADR standby database.

Delayed replay intentionally keeps the standby database at a point in time that is earlier than that of the primary database by delaying replay of logs on that standby. If an errant transaction is executed on the primary, you have until the configured time delay has elapsed to take action to prevent the errant transaction from being replayed on the standby. To recover the lost data, you can either copy this data back to the primary, or you can have the standby take over as the new primary database.

Delayed replay works by comparing timestamps in the log stream, which is generated on the primary, and the current time of the standby. As a result, it is important to synchronize the clocks of the primary and standby databases. Transaction commit is replayed on the standby according to the following equation:

*(current time on the standby - value of the **hadr\_replay\_delay** configuration parameter) >= timestamp of the committed log record*

You should set the **hadr\_replay\_delay** database configuration parameter to a large enough value to allow time to detect and react to errant transactions on the primary.

You can use this feature with one standby, multiple standbys, and in a Db2 pureScale environment. With multiple standbys, typically one or more standbys stays current with the primary for high availability or disaster recovery purposes, and one standby is configured with delayed replay for protection against errant transactions. If you use this feature with one standby, you should not enable IBM Tivoli System Automation for Multiplatforms because the takeover will fail.

There are several important restrictions for delayed replay:

- You can set the **hadr\_replay\_delay** configuration parameter only on a standby database.

- A **TAKEOVER** command on a standby with replay delay enabled fails. You must first set the **hadr\_replay\_delay** configuration parameter to 0 and then deactivate and reactivate the standby to pick up the new value, and then issue the **TAKEOVER** command.
- The delayed replay feature is supported only in SUPERASYNC mode. Because log replay is delayed, numerous non-replayed log data might accumulate on the standby, filling up receive buffer and spool (if configured). In other synchronization modes, this would cause the primary to be blocked.  
The objective of this feature is to protect against application error. If you want to use this feature and ensure that there is no data loss in the event of a primary failure, consider a multiple standby setup with a more synchronous setting on the principal standby.
- When you upgrade HADR databases, an important verification step for upgrade is to ensure that the primary's log shipping position matches the standby's log replay position. Naturally, a standby with replay delay configured can interfere with this verification step and cause it to fail. To avoid any failures, you must first set the **hadr\_replay\_delay** configuration parameter to 0, deactivate and reactivate the standby database to pick up the new value, and then start the upgrade procedure.

## Recommendations

### Delayed replay and disaster recovery

Consider using a small delay if you are using the standby database for disaster recovery purposes and errant transaction protection.

### Delayed replay and the HADR reads on standby feature

Consider using a small delay if you are using the standby database for reads on standby purposes, so that reader sessions can see more up-to-date data. Additionally, because reads on standby runs in “uncommitted read” isolation level, it can see applied, but not yet committed changes that are technically still delayed from replay. These uncommitted transactions can be rolled back in errant transaction recovery procedure when you roll forward the standby to the PIT that you want and then stop.

### Delayed replay and log spooling

If you enable delayed replay, it is recommended that you also enable log spooling by setting the **hadr\_spool\_limit** database configuration parameter. Because of the intentional delay, the replay position can be far behind the log receive position on the standby. Without spooling, log receive can only go beyond replay by the amount of the receive buffer. With spooling, the standby can receive many more logs beyond the replay position, providing more protection against data loss in case of primary failure. Note that in either case, because of the mandatory SUPERASYNC mode, the primary will not be blocked by the delayed replay.

## Recovering data by using HADR delayed replay

Using the HADR time-delayed replay feature, you can recover data that was lost because of an errant transaction on the primary database by stopping HADR on a standby before that transaction is replayed.

## Before you begin

Delayed replay must have already been enabled for your standby database.

If log replay on the standby, indicated by **STANDBY\_REPLAY\_LOG\_TIME**, has passed the commit time for the errant transaction on the standby, you cannot

recover the data using the following procedure. You can determine the `STANDBY_REPLAY_LOG_TIME` by using the `db2pd` command with the `-hadr` parameter or the `MON_GET_HADR` table function.

**Restriction:** A standby database for which you set the `hadr_replay_delay` configuration parameter cannot take over as a primary; you must first disable delayed replay on that standby.

## Procedure

To recover from an errant transaction, perform the following steps on the standby on which you enabled delayed replay:

1. Verify the timing:
  - a. Ensure that standby has not yet replayed the transaction. The `STANDBY_REPLAY_LOG_TIME` value must not have reached the errant transaction commit time.
  - b. Ensure that the standby has received the relevant logs. The `STANDBY_LOG_TIME` value, which indicates logs received, must have reached a PIT before the errant transaction commit time, but close to the errant transaction commit time. This will be the rollforward PIT used in step 3. If the standby has not yet received enough log files, you can wait until more logs are shipped over, but you run the risk of the replay time reaching the errant transaction time. For example, if the delay is 1 hour, you should stop HADR no later than 50 minutes after the errant transaction time (allowing a 10-minute safety margin), even if log shipping has yet not reached the PIT that you want.

Alternatively, if a shared log archive is available and the logs are already archived, then there is no need to wait. If the logs are not archived yet, the logs can be archived using the **ARCHIVE LOG** command. Otherwise, the user can manually copy complete log files from the primary to the time-delayed standby (the overflow log path is preferred, otherwise, use the log path). For these alternate methods, deactivate the standby first to avoid interference with standby log shipping and replay.

You can determine these times by issuing `db2pd -db dbname -hadr` or by enabling the reads on standby feature on the standby and then issuing the following query, which uses the `MON_GET_HADR` table function:

```
DB2 "select HADR_ROLE, STANDBY_ID, STANDBY_LOG_TIME, STANDBY_REPLAY_LOG_TIME,
varchar(PRIMARY_MEMBER_HOST,20) as PRIMARY_MEMBER_HOST,
varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST
from table (mon_get_hadr(NULL))"
```

2. Stop HADR on the standby database:  
`DB2 STOP HADR ON DATABASE dbname`
3. Roll forward the standby to the PIT that you want and then stop:  
`DB2 ROLLFORWARD DB dbname to time-stamp and STOP`
4. Use one of the following approaches:
  - Restore the lost data on the primary:
    - a. Copy the affected data from the standby and send it back to the primary.  
If the errant transaction dropped a table, you could export it on the standby and import it to the primary. If the errant transaction deleted rows from a table, you could export the table on the standby and use an import replace operation on the primary.



- b. Reinitialize the delayed-replay standby because its log stream has diverged from the primary's. No action is needed on any other standbys because they continue to follow the primary and any data repair on the primary is also replicated to them.
- c. Restore the database using a backup image taken on the primary. The image can be one taken at any time.
- d. Remove all log files in standby log path. This step is important. The **ROLLFORWARD... STOP** command in step 3 made the database log stream diverge from the primary. If the files are left alone, the newly restored database would follow that log stream and also diverge from the primary. Alternatively, you can drop the database before the restore for a clean start, but then you will also lose the current configuration including HADR configuration.
- e. Issue the **START HADR** command with the AS STANDBY option on the database. The database should then activate and connect to the primary.
- Have the standby with the intact data become the primary:
  - a. Shut down the old primary to avoid split brain
  - b. On the delayed-replay database, set the **hadr\_replay\_delay** configuration parameter to 0. Reconfigure the other parameters like **hadr\_target\_list** if needed. Then run **START HADR** command with the AS PRIMARY BY FORCE options on the database to convert it to the new primary. Use the BY FORCE option because there is no guarantee that the configured principal standby (which could be the old primary) will be able to connect.
  - c. Redirect clients to the new primary.
  - d. The other standbys will be automatically redirected to the new primary. However, if a standby received logs from the old primary beyond the point where old and new primary diverge (the PIT used in step 3), it will be rejected by the new primary. If this happens, reinitialize this standby using the same procedure as reinitializing the old primary.
  - e. Reinitialize the old primary because its log stream has diverged from the new primary's.
  - f. Restore database using a backup image taken on the new primary, or taken on the old primary before the PIT used in step 3.
  - g. Remove all log files in the log path. If you do not do this, the newly restored database will follow the old primary's log stream and diverge from the new primary. Alternatively, you can drop the database before the restore for a clean start, but then you also lose the current configuration including HADR configuration.
  - h. Issue the **START HADR** command with the AS STANDBY option on the database. The database should then activate and connect to the primary.

## Db2 High availability disaster recovery (HADR) management

Db2 High availability disaster recovery (HADR) management involves configuring and maintaining the status of your HADR system.

Managing HADR includes such tasks as:

- Cataloging an HADR database.
- “Initializing high availability disaster recovery (HADR)” on page 110
- Checking or altering database configuration parameters related to HADR.
- “Switching database roles in high availability disaster recovery (HADR)” on page 251

- “Performing an HADR failover operation” on page 248
- “High availability disaster recovery (HADR) monitoring” on page 245
- “Stopping Db2 High Availability Disaster Recovery (HADR)” on page 176

You can manage HADR using the following methods:

- Command line processor
- Db2 administrative API
- Task assistants for managing HADR in IBM Data Studio Version 3.1 or later.

#### Related information:

 Administering databases with task assistants

## Db2 High Availability Disaster Recovery (HADR) commands

The Db2 High Availability Disaster Recovery (HADR) feature provides complex logging, failover, and recovery functionality for Db2 high availability database solutions.

Despite the complexity of the functionality HADR provides, there are only a few actions you need to directly command HADR to perform: starting HADR; stopping HADR; and causing the standby database to take over as the primary database.

There are three high availability disaster recover (HADR) commands used to manage HADR:

- **START HADR**
- **STOP HADR**
- **TAKEOVER HADR**

To invoke these commands, use the command line processor or the administrative API.

Issuing the **START HADR** command with either the AS PRIMARY or AS STANDBY option changes the database role to the one specified if the database is not already in that role. This command also activates the database, if it is not already activated.

The **STOP HADR** command changes an HADR database (either primary or standby) into a standard database. Any database configuration parameters related to HADR remain unchanged so that the database can easily be reactivated as an HADR database.

The **TAKEOVER HADR** command, which you can issue on the standby database only, changes the standby database to a primary database. When you do not specify the BY FORCE option, the primary and standby databases switch roles. When you do specify the BY FORCE option, the standby database unilaterally switches to become the primary database. In this case, the standby database attempts to stop transaction processing on the old primary database. However, there is no guarantee that transaction processing will stop. Use the BY FORCE option to force a takeover operation for failover conditions only. To whatever extent possible, ensure that the current primary has definitely failed, or shut it down yourself, prior to issuing the **TAKEOVER HADR** command with the BY FORCE option.

## HADR database role switching

A database can be switched between primary and standard roles dynamically and repeatedly. When the database is either online or offline, you can issue both the **START HADR** command with the AS PRIMARY option and the **STOP HADR** command.

You can switch a database between standby and standard roles statically. You can do so repeatedly only if the database remains in rollforward pending state. You can issue the **START HADR** command with the AS STANDBY option to change a standard database to standby while the database is offline and in rollforward pending state. Use the **STOP HADR** command to change a standby database to a standard database while the database is offline. The database remains in rollforward pending state after you issue the **STOP HADR** command. Issuing a subsequent **START HADR** command with the AS STANDBY option returns the database to standby. If you issue the **ROLLFORWARD DATABASE** command with the STOP option after stopping HADR on a standby database, you cannot bring it back to standby. Because the database is out of rollforward pending state, you can use it as a standard database. This is referred to as taking a snapshot of the standby database. After changing an existing standby database into a standard database, consider creating a new standby database for high availability purposes.

To switch the role of the primary and standby databases, perform a takeover operation without using the BY FORCE option.

To change the standby to primary unilaterally (without changing the primary to standby), use forced takeover. Subsequently, you might be able to reintegrate the old primary as a new standby.

HADR role is persistent. Once an HADR role is established, it remains with the database, even through repeated stopping and restarting of the Db2 instance or deactivation and activation of the Db2 database.

### **Starting the standby is asynchronous**

When you issue the **START HADR** command with the AS STANDBY option, the command returns as soon as the relevant engine dispatchable units (EDUs) are successfully started. The command does not wait for the standby to connect to the primary database. In contrast, the primary database is not considered started until it connects to a standby database (with the exception of when the **START HADR** command is issued on the primary with the BY FORCE option). If the standby database encounters an error, such as the connection being rejected by the primary database, the **START HADR** command with the AS STANDBY option might have already returned successfully. As a result, there is no user prompt to which HADR can return an error indication. The HADR standby will write a message to the Db2 diagnostic log and shut itself down. You should monitor the status of the HADR standby to ensure that it successfully connects with the HADR primary.

Replay errors, which are errors that the standby encounters while replaying log records, can also bring down the standby database. These errors might occur, for example, when there is not enough memory to create a buffer pool, or if the path is not found while creating a table space. You should continuously monitor the status of the standby database.

### **Do not run HADR commands from a client using a database alias enabled for client reroute**

When automatic client reroute is set up, the database server has a predefined alternate server so that client applications can switch between working with either the original database server or the alternative server with only minimal interruption of the work. In such an environment, when a client connects to the database via TCP, the actual connection can go to either the original database or to the alternate database. HADR commands are implemented to identify the target

database through regular client connection logic. Consequently, if the target database has an alternative database defined, it is difficult to determine the database on which the command is actually operating. Although an SQL client does not need to know which database it is connecting to, HADR commands must be applied on a specific database. To accommodate this limitation, HADR commands should be issued locally on the server machine so that client reroute is bypassed (client reroute affects only TCP/IP connections).

## HADR multiple standby databases

High availability disaster recovery (HADR) supports multiple standby databases. Using multiple standbys, you can have your data in more than two sites, which provides improved data protection with a single technology.

HADR allows you to have up to three standby databases in your setup. You designate one of these databases as the *principal HADR standby database*; any other standby database is an *auxiliary HADR standby database*. Both types of HADR standbys are synchronized with the HADR primary database through a direct TCP/IP connection, both types support reads on standby, and you can configure both types for time-delayed log replay. In addition, you can issue a forced or non-forced takeover on any standby. There are a couple of important distinctions between the principal and auxiliary standbys, however:

- IBM Tivoli System Automation for Multiplatforms (SA MP) automated failover is supported only for the principal standby. You must issue a takeover manually on one of the auxiliary standbys to make one of them the primary. Before issuing a manual takeover, you should disable SA MP.
- All of the HADR synchronization modes are supported on the principal standby, but the auxiliary standbys can only be in SUPERASYNC mode.

There are a number of benefits to using a multiple HADR standby setup. Instead of employing the HADR feature to achieve your high availability objectives and another technology to achieve your disaster recovery objectives, you can use HADR for both. You can deploy your principal standby in the same location as the primary. If there is an outage on the primary, the principal standby can take over the primary role within your recovery time objectives. You can also deploy auxiliary standbys in a distant location, which provides protection against a widespread disaster that affects both the primary and the principal standby. The distance, and the potential for network delays between the primary and the auxiliaries, has no effect on activity on the primary because the auxiliaries use SUPERASYNC mode. If a disaster affects the primary and principal standby, you can issue a takeover on either of the auxiliaries. You can configure the other auxiliary standby database to become the new principal standby using the **hadr\_target\_list** database configuration parameter. However, an auxiliary standby can take over as the primary even if that auxiliary does not have an available standby. For example, if there is an outage on the primary and principal standby, one auxiliary can take over as the primary even if it does not have a corresponding standby. However, if you stop that database after it becomes the new primary, it cannot start again as an HADR primary unless its principal standby is started.

### Restrictions for multiple standby databases

There are a number of restrictions that you should be aware of if you are planning to deploy multiple HADR standby databases.

The restrictions are as follows:

- You can have a maximum of three standby databases: one principal standby and up to two auxiliary standbys. The exception is for Db2 pureScale environments, in which you can only have one standby (the principal standby).
- Only the principal standby supports all the HADR synchronization modes; all auxiliary standbys will be in SUPERASYNC mode. The exception is for Db2 pureScale environments, in which the principal standby can only be in ASYNC or SUPERASYNC mode.
- IBM Tivoli System Automation for Multiplatforms (SA MP) support applies only between the primary HADR database and its principal standby. The exception is for Db2 pureScale environments, in which SA MP only manages the local cluster and does not automate failover to the standby cluster
- The **hadr\_target\_list** database configuration parameter must be set on all the databases in the multiple standby setup. Each standby must include the primary in its **hadr\_target\_list** setting.

## Modifications to a multiple standby database setup

After your multiple HADR standby setup is up and running, you might want to make additional changes, such as adding or removing auxiliary standby databases or changing the principal standby database designation. You can make these kinds of modifications without causing an outage on your primary database.

### Adding auxiliary standbys

There are a few reasons why you might want to add an auxiliary standby:

- To deploy an additional standby for processing read-only workloads
- To deploy an additional standby for time-delayed replay
- To deploy an additional standby for disaster recovery purposes
- To add a standby that was a part of a previously active HADR deployment but was *orphaned* because the **hadr\_target\_list** configuration parameter for the new primary does not specify that standby

You can add an auxiliary standby only if the **hadr\_target\_list** configuration parameter is already set on the primary and at least one standby.

To add an auxiliary standby to your HADR deployment, update the target list of the primary with the host and port information from the standby. This information corresponds to the settings for the **hadr\_local\_host** and **hadr\_local\_svc** parameters on the standby. You must also add the host and port information for the primary to the target list of the new standby.

**Tip:** Although it is not required, a best practice is to also add the host and port information for the new standby to the target lists of the other standbys in the deployment. You should also specify the host and port information for those standbys in the target list of the new standby. If you do not make these additional updates and one of the other standbys takes over as the new primary, the new standby is rejected as a standby target and is shut down.

### Removing auxiliary standbys

The only standbys that you can remove dynamically are auxiliary standbys. If you dynamically remove an auxiliary standby from your multiple standby deployment, there is no effect on normal HADR operations on the primary and the principal standby. To remove an auxiliary standby, issue the **STOP HADR** command on the standby; afterward, you can remove it from the target lists of the primary and any other standby.

## Changing the principal standby

You can change the principal standby only if you first stop HADR on the primary database; this does not cause an outage, because you do not have to deactivate the primary.

To change the principal standby, you must stop HADR on the primary database. Then, update the target list of the primary database to list the new principal standby first. If the new principal standby is not already a standby, add the primary database's address to its target list, configure the other HADR parameters, and activate the standby. If it is already a standby, no action is needed.

**Tip:** Although it is not required, it is a best practice to also add the host and port information for the new principal standby to the target list of the other standby in the deployment. You should also specify the host and port information for that standby in the target list of the new principal standby. If you do not make these additional updates and either one of the standbys takes over as the new primary, the other standby is rejected as a standby target and is shut down.

## Database configuration for multiple HADR standby databases

There are a number of considerations for database configuration in a multiple HADR standby setup.

## Automatic reconfiguration of HADR parameters

### Reconfiguration after HADR starts

The configuration parameters that identify the primary database for the standbys and identify the principal standby for the primary are automatically reset when HADR starts if you did not correctly set them. This behavior applies to the following configuration parameters:

- **hadr\_remote\_host**
- **hadr\_remote\_inst**
- **hadr\_remote\_svc**

**Tip:** Even though this automatic reconfiguration occurs, you should always try to set the correct initial values because that reconfiguration might not take effect until a connection is made between a standby and its primary. In some HADR deployments, those initial values might be needed. For example, if you are using the IBM Tivoli System Automation for Multiplatforms software, the value for the **hadr\_remote\_inst** configuration parameter is needed to construct a resource name.

**Note:** If the **DB2\_HADR\_NO\_IP\_CHECK** registry variable is set to ON, the **hadr\_remote\_host** and **hadr\_remote\_svc** are not automatically updated.

Reconfiguration is predicated on the values of the **hadr\_target\_list** configuration parameter being correct; if anything is wrong in a target list entry, you must correct it manually.

On the primary, the reconfiguration occurs in the following manner:

- If the values for the **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters do not match the *host:port* pair that is the first entry of the **hadr\_target\_list** configuration parameter (namely, the principal standby), the **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters are updated with the values from the target list.

- If the value for the **hadr\_remote\_inst** configuration parameter does not match the instance name of the principal standby, the correct instance name is copied to the **hadr\_remote\_inst** configuration parameter for the primary after the principal standby connects to it.

On a standby database, the reconfiguration occurs in the following manner:

- When a standby starts, it attempts to connect to the database that you specified for its **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters.
- If the standby cannot connect to the primary, it waits for the primary to connect to it.
- The primary attempts to connect to its standbys using addresses listed in its **hadr\_target\_list** parameter. After the primary connects to a standby, the **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters for the standby are updated with the correct values for the primary.

### Reconfiguration during and after a takeover

In a non-forced takeover, the values for the **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters on the new primary are automatically updated to its principal standby, and these parameters on the standbys listed in the new primary's **hadr\_target\_list** are automatically updated to point to the new primary. Any database that is not listed in the new primary's **hadr\_target\_list** is not updated. Those databases continue to attempt to connect to the old primary and get rejected because the old primary is now a standby. The old primary is guaranteed to be in the new primary's target list because of the requirement of mutual inclusion in the target list.

In a forced takeover, automatic update on the new primary and its standbys (excluding the old primary) work the same way as non-forced takeover. However, automatic update on the old primary does not happen until it is shut down and restarted as a standby for reintegration.

Any database that is not online during the takeover will be automatically reconfigured after it starts. Automatic reconfiguration might not take effect immediately on startup, because it relies on the new primary to periodically contact the standby. On startup, a standby might attempt to connect to the old primary and follow the log stream of the old primary, causing it to diverge from the new primary's log stream and, making that standby unable to pair with the new primary. As a result, you must shut down the old primary before takeover to avoid that kind of *split brain* scenario.

### Lack of standby control of the synchronization mode and peer window

With multiple standbys, only the settings of the **hadr\_syncmode** and **hadr\_peer\_window** configuration parameters of the current primary are relevant. The standby databases either have the settings for those parameters defined by the primary, in the case of the principal standby, or by their role as an auxiliary standby.

#### Synchronization mode

The setting for the **hadr\_syncmode** configuration parameter does not have to be the same on the primary and standby databases. Whatever setting you specify for the **hadr\_syncmode** configuration parameter on a standby is considered its *configured synchronization mode*; this setting has relevance

only if the standby becomes a primary. The standby is assigned an *effective synchronization mode*. For any auxiliary standby, the effective synchronization mode is always SUPERASYNC. For the principal standby, the effective synchronization mode is the setting for the **hadr\_syncmode** configuration parameter for the primary. For a standby, the monitoring interfaces display the effective synchronization mode as the synchronization mode.

#### Peer window

The setting for the **hadr\_peer\_window** configuration parameter does not have to be the same on the primary and standby databases. In fact, any setting for the **hadr\_peer\_window** configuration parameter on the auxiliary standbys is ignored because peer window functionality is incompatible with SUPERASYNC mode. The principal standby uses the peer window setting of the primary, which is applicable only if the value of the **hadr\_syncmode** configuration parameter for the standby is SYNC or NEARSYNC.

### Rolling updates with multiple HADR standby databases

Multiple HADR standbys allow you to use the rolling update procedure while maintaining HADR protection by keeping a primary and a standby active.

There is always a primary to provide database service and this primary always has at least one standby providing HA and DR protection.

You should perform the update or upgrade on all of the standbys before doing so on the primary. This is particularly important if you are updating the fixpack level because HADR does not allow the primary to be at a higher fixpack level than the standby.

The procedure is essentially the same as with one standby, except you should perform the upgrade on one database at a time and starting with an auxiliary standby. For example, consider the following HADR setup:

- host1 is the primary
- host2 is the principal standby
- host 3 is the auxiliary standby

For this setup, perform the rolling upgrade or update according to the following sequence:

1. Deactivate host3, make the required changes, activate host3, and start HADR on host3 (as a standby).
2. After host3 is caught up in log replay, deactivate host2, make the required changes, activate host2, and start HADR on host2 (as a standby).
3. After host2 is caught up in log replay and in peer state with host1, issue a takeover on host2.
4. Deactivate host1, make the required changes, activate host1, and start HADR on host1 (as a standby).
5. After host1 is in peer state with host 2, issue a takeover on host1 so that it becomes the primary again and host2 becomes the principal standby again.

### High availability disaster recovery (HADR) monitoring for multiple standby databases

When monitoring multiple HADR standby databases, you should only use the **db2pd** command and the **MON\_GET\_HADR** table function because other monitoring interfaces do not give a complete view of all of the standbys.



The information returned by the monitoring interface depends on where it is issued. Monitoring on a standby returns information about that standby and the primary only; no information is provided about any other standbys. Monitoring on the primary returns information about all of the standbys if you are using the **db2pd** command or the MON\_GET\_HADR table function. Even standbys that are not connected, but are configured in the primary's **hadr\_target\_list** configuration parameter are displayed. Other interfaces like the **GET SNAPSHOT FOR DATABASE** command, the SNAPHADR administrative view, and the SNAP\_GET\_HADR table function, report the primary and the principal standby only; these interfaces have been deprecated and might be removed in a future release.

The **db2pd** command and the MON\_GET\_HADR table function return essentially the same information, but the **db2pd** command does not require reads on standby to be enabled (for reporting from a standby). As well, the **db2pd** command is preferred during takeover because there could be a time window where neither the primary nor the standby allows client connections.

## db2pd command

In the following example, the DBA issues the **db2pd** command on a primary database with three standbys. Three sets of data are returned, with each representing a primary-standby log shipping channel. The HADR\_ROLE field represents the role of the database to which **db2pd** is issued, so it is listed as PRIMARY in all sets. The HADR\_STATE for the two auxiliary standbys (hostS2 and hostS3) is REMOTE\_CATCHUP because they automatically run in SUPERASYNC mode (which is also reflected in the **db2pd** output) regardless of their configured setting for **hadr\_syncmode**. The STANDBY\_ID differentiates the standbys. It is system generated and the ID-to-standby mapping can change from query to query; however, the ID "1" is always assigned to the principal standby.

**Note:** Fields not relevant to current status might be omitted in the output. For example, in the following output, information about the replay-only window (like start time and transaction count) is not included because the replay-only window is not active.

```
db2pd -db hadr_db -hadr
```

```
Database Member 0 -- Database hadr_db -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23
```

```

HADR_ROLE = PRIMARY
REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = SYNC
STANDBY_ID = 1
LOG_STREAM_ID = 0
HADR_STATE = PEER
HADR_FLAGS =
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS1.ibm.com
STANDBY_INSTANCE = db2inst2
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 3
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82

```

```

SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
    PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
    STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        HADR_LOG_GAP(bytes) = 0
    STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
    STANDBY_RECV_REPLAY_GAP(bytes) = 0
        PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
    STANDBY_RECV_BUF_SIZE(pages) = 16
    STANDBY_RECV_BUF_PERCENT = 0
    STANDBY_SPOOL_LIMIT(pages) = 0
    STANDBY_SPOOL_PERCENT = 0
    PEER_WINDOW(seconds) = 0
    READS_ON_STANDBY_ENABLED = Y
    STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

        HADR_ROLE = PRIMARY
        REPLAY_TYPE = PHYSICAL
        HADR_SYNCMODE = SUPERASYNC
        STANDBY_ID = 2
        LOG_STREAM_ID = 0
        HADR_STATE = REMOTE_CATCHUP
    PRIMARY_MEMBER_HOST = hostP.ibm.com
    PRIMARY_INSTANCE = db2inst1
    PRIMARY_MEMBER = 0
    STANDBY_MEMBER_HOST = hostS2.ibm.com
    STANDBY_INSTANCE = db2ins3t
    STANDBY_MEMBER = 0
    HADR_CONNECT_STATUS = CONNECTED
    HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
    HEARTBEAT_INTERVAL(seconds) = 30
    HADR_TIMEOUT(seconds) = 120
    TIME_SINCE_LAST_RECV(seconds) = 16
    PEER_WAIT_LIMIT(seconds) = 0
    LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
    LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
    LOG_HADR_WAIT_COUNT = 82
    SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
    SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
        PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            HADR_LOG_GAP(bytes) = 0
        STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        STANDBY_RECV_REPLAY_GAP(bytes) = 0
            PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_RECV_BUF_SIZE(pages) = 16
        STANDBY_SPOOL_LIMIT(pages) = 0
        STANDBY_SPOOL_PERCENT = 0
        PEER_WINDOW(seconds) = 0
        READS_ON_STANDBY_ENABLED = Y
        STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

            HADR_ROLE = PRIMARY
            REPLAY_TYPE = PHYSICAL
            HADR_SYNCMODE = SUPERASYNC
            STANDBY_ID = 3
            LOG_STREAM_ID = 0
            HADR_STATE = REMOTE_CATCHUP
        PRIMARY_MEMBER_HOST = hostP.ibm.com
        PRIMARY_INSTANCE = db2inst1
        PRIMARY_MEMBER = 0
        STANDBY_MEMBER_HOST = hostS3.ibm.com
        STANDBY_INSTANCE = db2inst3
        STANDBY_MEMBER = 0
        HADR_CONNECT_STATUS = CONNECTED
        HADR_CONNECT_STATUS_TIME = 06/08/2011 13:46:51.561873 (1307566011)

```

```

HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 6
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
STANDBY_SPOOL_PERCENT = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

```

## MON\_GET\_HADR table function

In the following example, the DBA calls the **MON\_GET\_HADR** table function on the primary database with three standbys. Three rows are returned. Each row represents a primary-standby log shipping channel. The **HADR\_ROLE** column represents the role of the database to which the query is issued. Therefore it is **PRIMARY** on all rows. The **HADR\_STATE** for the two auxiliary standbys (hostS2 and hostS3) is **REMOTE\_CATCHUP** because they automatically run in **SUPERASYNC** mode regardless of their configured setting for **hadr\_syncmode**.

```

db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST,20)
as PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST,20)
as STANDBY_MEMBER_HOST from table (mon_get_hadr(NULL))"

```

HADR_ROLE	STANDBY_ID	HADR_STATE	PRIMARY_MEMBER_HOST	STANDBY_MEMBER_HOST
PRIMARY	1	PEER	hostP.ibm.com	hostS1.ibm.com
PRIMARY	2	REMOTE_CATCHUP	hostP.ibm.com	hostS2.ibm.com
PRIMARY	3	REMOTE_CATCHUP	hostP.ibm.com	hostS3.ibm.com

3 record(s) selected.

## HADR takeover operations with multiple standbys

When an HADR standby database takes over as the primary database in a multiple standby setup, there are a number of important differences from when there is a single standby.

With HADR, there are two types of takeover: *role switch* and *failover*. Role switch, sometimes called graceful takeover or non-forced takeover, can be performed only when the primary is available and it switches the role of primary and standby. Failover, or forced takeover, can be performed when the primary is not available. It is commonly used in primary failure cases to make the standby the new primary. The old primary remains in the primary role in a forced takeover, but the standby sends it a message to disable it. Both types of takeover are supported with multiple standby databases, and any of the standby databases can take over as the primary. A crucial thing to remember, though, is that if a standby is not included in the new primary's target list, it is considered to be orphaned and cannot connect to the new primary.

In a takeover, a number of configuration changes are automatically made for you so that the standbys listed in new primary's target list can connect to the new primary. The **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters are updated on the new primary and listed standbys in the following way:

- **On the new primary:** They refer to the principal standby (the first database listed in the new primary's target list).
- **On the standbys:** They refer to the new primary. When an old primary is reintegrated to become standby, the **START HADR AS STANDBY** command first converts it to a standby. Thus it can also be automatically redirected to the new primary if it is listed in the target list of the new primary.

**Note:** Orphaned standbys are not automatically updated in this way. If you want them to join as standbys, you need to ensure they are in the new primary's target list and that they include the new primary in their target lists.

## Role switch

Just as in single standby mode, role switch in multiple standby mode guarantees no data is lost between the old primary and new primary. Other standbys configured in the new primary's **hadr\_target\_list** configuration parameter are automatically redirected to the new primary and continue receiving logs. If you are issuing the **TAKEOVER HADR** command on an auxiliary standby and you have IBM Tivoli System Automation for Multiplatforms (SA MP) configured, you must ensure that you disable SA MP before attempting the takeover. You cannot failback the primary role to the auxiliary primary if SA MP is enabled.

## Failover

Just as with one standby, if a failover results in any data loss with multiple standbys (meaning that the new primary does not have all of the data of the old primary), the old and new primary's log streams diverge and the old primary has to be reinitialized. For the other standbys, if a standby received logs from the old primary beyond the diverge point, it has to be reinitialized. Otherwise, it can connect to the new primary and continue log shipping and replay. As a result, it is very important that you check the log positions of all of the standbys and choose the standby with the most data as the failover target. You can query this information using the **db2pd** command or the **MON\_GET\_HADR** table function.

**Note:** Successful automatic reconfiguration of a standby's **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters to point to the new primary does not mean the standby will be accepted to pair with the new primary. It only allows the standby to make a TCP connection to the primary. Upon connection, if Db2 determines that the two databases have diverging log streams, the pairing request will be rejected and the connection closed.

## Scenario: Deploying an HADR multiple standby database setup

This scenario describes the planning, configuring, and deploying of an HADR setup for a bank called ExampleBANK. The setup has three standby databases: one principal standby and two auxiliary standbys.

## Background

Because banking is a 24x7 business, high availability is crucial to ExampleBANK's technology strategy. In addition, ExampleBANK experienced a close call with a hurricane hitting City A, where its head office is located, so the bank also requires

a disaster recovery strategy. High availability disaster recovery (HADR) offers a solution that can help the bank achieve both of these goals with a single technology: HADR multiple standby databases.

ExampleBANK considers the following requirements essential for its HADR solution:

**An aggressive recovery time objective**

As a bank that offers 24-hour online service, ExampleBANK wants to minimize the time that applications cannot connect to their database.

**An aggressive recovery point objective**

ExampleBANK cannot tolerate data loss, so the RPO should be as close to 0 as possible.

**Near-zero planned downtime**

ExampleBANK's database should be available as much as possible, even through planned activities such as upgrades and maintenance.

**Data protection through geographic dispersion**

As part of its compliance standards, ExampleBANK wants the capability to recover operations at a remote location.

**Easy deployment and management**

ExampleBANK's overburdened IT department wants a solution that is relatively simple to configure and that has automation capabilities.

As the following scenarios illustrate, using the HADR feature with multiple standby databases helps ExampleBANK meet all these requirements.

## **Planning for a multiple standby setup**

ExampleBANK wants to have both high availability and disaster recovery protection from its HADR setup, so the bank decides to use the maximum number of standbys: three. To achieve the RTO, the bank must have a standby that is in close synchronization with the primary (a standby that uses SYNC or NEARSYNC mode) and is collocated with the primary. It makes the most sense to have this standby be the principal standby because only that standby supports all synchronization modes. Both the primary and the principal standby are located in ExampleBANK's head office in City A and are connected by a LAN.

In addition, to protect the bank's data from being lost because of a disaster, the ExampleBANK DBA chooses to set up two standbys in the bank's regional office in City B. The regional office is connected to the head office in City A by a WAN. The distance between the two cities will not affect the primary because the standbys are auxiliary standbys, which automatically run in SUPERASYNC mode. The DBA can provide additional justification for the costs of these additional databases by setting up one of them to use the reads on standby feature and the other to use the time-delayed replay feature. Also, these standbys can help maintain database availability through a rolling update or maintenance scenario, preventing the loss of HADR protection.

## **Configuring a multiple standby setup**

The ExampleBANK DBA takes a backup of the intended primary database, HADR\_DB:

```
DB2 BACKUP DB hadr_db TO backup_dir
```

The DBA then restores the backup onto each of the intended standby hosts by issuing the following command:

```
DB2 RESTORE DB hadr_db FROM backup_dir
```

**Tip:** For more information about options for creating a standby, see “Initializing a standby database” on page 109.

For the initial setup, the ExampleBANK DBA decides that most of the default configuration settings are sufficient. However, as in a regular HADR setup, the following database configuration parameters must be explicitly set:

- **hadr\_local\_host**
- **hadr\_local\_svc**
- **hadr\_remote\_host**
- **hadr\_remote\_inst**
- **hadr\_remote\_svc**

To obtain the correct values for those configuration parameters, the DBA determines the host name, port number, and instance name of the four databases that will be in the HADR setup:

*Table 11. Host name, port number, and instance name for databases*

Intended role	Host name	Port number	Instance name
Primary	host1	10	dbinst1
Principal standby	host2	40	dbinst2
Auxiliary standby	host3	41	dbinst3
Auxiliary standby	host4.ibm.com	42	dbinst4

On the primary, the settings for the **hadr\_remote\_host**, **hadr\_remote\_inst**, and **hadr\_remote\_svc** configuration parameters correspond to the host name, instance name, and port number of the principal standby. On the standbys, the values of these configuration parameters correspond to the host name, port number, and instance name of the primary. In addition, the DBA uses the host name and port values to set the **hadr\_target\_list** configuration parameter on all the databases. Also, although it is not required, the DBA adds the information about all the standbys in the setup to the target list of each of the other standbys. For more information about this topic, see “Database configuration for high availability disaster recovery (HADR)” on page 117.

As mentioned earlier, the bank wants the closest possible synchronization between the primary and principal standby, so the DBA sets the **hadr\_syncmode** parameter on the primary to SYNC. Although the principal standby will automatically have its effective synchronization mode set to SYNC after it connects to the primary, the DBA still sets the **hadr\_syncmode** parameter to SYNC on the principal standby. The reason is that if the principal standby switches role with the primary, the synchronization mode for the new primary and principal standby pair will also be SYNC.

The DBA decides to specify host2, which is in a different city from the auxiliary standbys, as the principal standby for the auxiliary standbys. If one of the auxiliaries becomes the primary, SUPERASYNC would be a good synchronization mode between the primary and the remotely located host2. Thus DBA sets the **hadr\_syncmode** parameter on the auxiliary standbys to SUPERASYNC, although the auxiliary standbys will automatically have their effective synchronization modes

set to SUPERASYNC after they connect to the primary. For more information about this topic, see “High availability disaster recovery (HADR) synchronization mode” on page 134.

Finally, the DBA has read about the new HADR delayed replay feature, which can be used to intentionally keep a standby database at a point in time that is earlier than the primary by delaying replay of logs. The DBA decides that enabling this feature would improve ExampleBANK's data protection against errant transactions on the primary. The DBA chooses host4, an auxiliary standby, for this feature, and makes a note that this feature must be disabled before host4 can take over as the primary database. For more information about this topic, see “HADR delayed replay” on page 194.

The DBA issues the following commands to update the configuration parameters on each of the databases:

- On host1 (the primary):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host2:40|host3:41|host4:42
      HADR_REMOTE_HOST host2
      HADR_REMOTE_SVC 40
      HADR_LOCAL_HOST host1
      HADR_LOCAL_SVC 10
      HADR_SYNCMODE sync
      HADR_REMOTE_INST db2inst2"
```
- On host2 (the principal standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host1:10|host3:41|host4:42
      HADR_REMOTE_HOST host1
      HADR_REMOTE_SVC 10
      HADR_LOCAL_HOST host2
      HADR_LOCAL_SVC 40
      HADR_SYNCMODE sync
      HADR_REMOTE_INST db2inst1"
```
- On host3 (an auxiliary standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host2:40|host1:10|host4:42
      HADR_REMOTE_HOST host1
      HADR_REMOTE_SVC 10
      HADR_LOCAL_HOST host3
      HADR_LOCAL_SVC 41
      HADR_SYNCMODE superasync
      HADR_REMOTE_INST db2inst1"
```
- On host4 (an auxiliary standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host2:40|host1:10|host3:41
      HADR_REMOTE_HOST host1
      HADR_REMOTE_SVC 10
      HADR_LOCAL_HOST host4
      HADR_LOCAL_SVC 42
      HADR_SYNCMODE superasync
      HADR_REMOTE_INST db2inst1
      HADR_REPLAY_DELAY 86400"
```

Finally, the ExampleBANK DBA wants to enable the HADR reads on standby feature for the following reasons:

- To make online changes to some of the HADR configuration parameters on the standbys
- To call the MON\_GET\_HADR table function on the standbys

- To divert some of the read-only workload from the primary

The DBA updates the registry variables on the standby databases by issuing the following commands on each of host2, host3, and host4:

```
DB2SET DB2_HADR_ROS=ON
DB2SET DB2_STANDBY_ISO=UR
```

## Starting the HADR databases

The DBA starts the standby databases first, by issuing the following command on each of host2, host3, and host 4:

```
DB2 START HADR ON DB hadr_db AS STANDBY
```

Next, the DBA starts HADR on the primary database, on host1:

```
DB2 START HADR ON DB hadr_db AS PRIMARY
```

To verify that HADR is up and running, the DBA queries the status of the databases from the primary on host1 by issuing the **db2pd** command, which returns information about all of the standbys:

```
db2pd -db hadr_db -hadr
```

```
Database Member 0 -- Database hadr_db -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23

      HADR_ROLE = PRIMARY
      REPLAY_TYPE = PHYSICAL
      HADR_SYNCMODE = SYNC
      STANDBY_ID = 1
      LOG_STREAM_ID = 0
      HADR_STATE = PEER
      HADR_FLAGS =
PRIMARY_MEMBER_HOST = host1
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = host2
STANDBY_INSTANCE = db2inst2
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 3
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
PRIMARY_LOG_FILE,PAGE,POS = S00000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S00000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S00000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
STANDBY_SPOOL_PERCENT = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

      HADR_ROLE = PRIMARY
```



```

REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = SUPERASYNC
STANDBY_ID = 2
LOG_STREAM_ID = 0
HADR_STATE = REMOTE_CATCHUP
HADR_FLAGS =
PRIMARY_MEMBER_HOST = host1
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = host3
STANDBY_INSTANCE = db2inst3
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 16
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
STANDBY_SPOOL_PERCENT = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

HADR_ROLE = PRIMARY
REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = SUPERASYNC
STANDBY_ID = 3
LOG_STREAM_ID = 0
HADR_STATE = REMOTE_CATCHUP
HADR_FLAGS =
PRIMARY_MEMBER_HOST = host1
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = host4
STANDBY_INSTANCE = db2inst4
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:46:51.561873 (1307566011)
HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 6
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)

```

```

STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
STANDBY_SPOOL_PERCENT = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

```

### Examples: Takeover in a multiple HADR standby setup

This set of examples of takeovers (both forced and unforced) with multiple HADR standbys is based on a three-standby setup. The purpose of these examples is to show how the automatic reconfiguration works in a takeover situation.

- “A principal standby takes over gracefully (role switch)” on page 215
- “An auxiliary standby takes over by force (failover)” on page 216
- “An auxiliary standby takes over by force (failover) in a SA MP environment” on page 217

The initial setup for each of the examples is as follows:

- a primary database (host1)
- a principal standby (host2)
- two auxiliary standbys (host3 and host4)

All of the databases are called `hadr_db`. The primary and principal standby have their synchronization mode set to `SYNC` and the standbys have theirs set to `SUPERASYNC`.

The configuration for each database is shown in Table 12.

*Table 12. Configuration values for each HADR database*

Configuration parameter	Host1	Host2	Host3	Host4
<b>hadr_target_list</b>	host2:40   host3:41   host4:42	host1:10   host3:41   host4:42	host2:40   host1:10   host4:42	host2:40   host1:10   host3:41
<b>hadr_remote_host</b>	host2	host1	host1	host1
<b>hadr_remote_svc</b>	40	10	10	10
<b>hadr_remote_inst</b>	dbinst2	dbinst1	dbinst1	dbinst1
<b>hadr_local_host</b>	host1	host2	host3	host4
<b>hadr_local_svc</b>	10	40	41	42
Configured <b>hadr_syncmode</b> (Refers to the explicitly set synchronization mode, which is used if the database becomes a primary)	SYNC	SYNC	SUPERASYNC	SUPERASYNC

Table 12. Configuration values for each HADR database (continued)

Configuration parameter	Host1	Host2	Host3	Host4
Effective <b>hadr_syncmode</b> (Refers to the synchronization mode that is used if the database is currently a standby)	n/a	SYNC	SUPERASYNC	SUPERASYNC

## A principal standby takes over gracefully (role switch)

The DBA performs a takeover on the principal standby by issuing the following command on host2:

```
DB2 TAKEOVER HADR ON DB hadr_db
```

After the takeover is completed successfully, host2 becomes the new primary and host1, which is the first entry in the **hadr\_target\_list** of host2 (as shown in Table 12 on page 214), becomes its principal standby. Their sync mode is SYNC mode because host2 is configured with an **hadr\_syncmode** of SYNC. The auxiliary standby targets, host3 and host4, have their **hadr\_remote\_host** and **hadr\_remote\_svc** pointing at the old primary, host1, but are automatically redirected to the new primary, host2. In this redirection, host3 and host4 update (persistently) their **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters. They reconnect to host2 as auxiliary standbys, and are told by host2 to use an effective synchronization mode of SUPERASYNC (regardless of what they have locally configured for **hadr\_syncmode**). They do not update their settings for **hadr\_syncmode** persistently. The configuration for each database is shown in Table 13.

Table 13. Configuration values for each HADR database after a role switch. Rows 3 to 5 in columns 4 and 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
<b>hadr_target_list</b>	host2:40   host3:41   host4:42	host1:10   host3:41   host4:42	host2:40   host1:10   host4:42	host2:40   host1:10   host3:41
<b>hadr_remote_host</b>	host2	host1	<b>host2</b>	<b>host2</b>
<b>hadr_remote_svc</b>	40	10	<b>40</b>	<b>40</b>
<b>hadr_remote_inst</b>	dbinst2	dbinst1	<b>dbinst2</b>	<b>dbinst2</b>
<b>hadr_local_host</b>	host1	host2	host3	host4
<b>hadr_local_svc</b>	10	40	41	42
Configured <b>hadr_syncmode</b>	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective <b>hadr_syncmode</b>	SYNC	n/a	SUPERASYNC	SUPERASYNC

**Note:** A number of values are not updated for the following reasons

- Because host2 already has its **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters pointing at its principal standby, host1, these values are not updated on host2.
- Because host1 already has its **hadr\_remote\_host** and **hadr\_remote\_svc** configuration parameters pointing at the new primary, these values are not updated on host1.
- Because host1's operational synchronization mode is SYNC and host3 and host4's operational synchronization modes are SUPERASYNC, there is no change for the effective synchronization mode.

## An auxiliary standby takes over by force (failover)

A widespread power outage in City A results in the primary (host1) becoming unavailable. Normally, the principal standby (host2) which is in SYNC mode would be the best candidate for taking over and becoming the new primary, but the power outage means that host2 is momentarily unavailable as well. The DBA queries the two auxiliary standbys to determine which one has the most log data:

```
db2pd -hadr -db hadr_db | grep 'PRIMARY_LOG_FILE,PAGE,POS|STANDBY_LOG_FILE,PAGE,POS'
```

The DBA determines that host3 is the most up to date (although it is still a little behind in log replay) and picks that host as the new primary:

```
DB2 TAKEOVER HADR ON DB hadr_db BY FORCE
```

After the takeover is completed successfully, host3 becomes the new primary. Meanwhile, host2 becomes available again. host3 informs host2 and host4 that it is now the primary. On host3, the values for **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** are reconfigured to point to host2, which is the principal standby because it is the first entry in the **hadr\_target\_list** on host3. On host2, the synchronization mode is reconfigured to SUPERASYNC because that is the setting for **hadr\_syncmode** on host3; in addition, the **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** are updated (persistently). host4 is automatically redirected to the new primary, host3. In this redirection, host4 updates (persistently) its **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters. There is no automatic reconfiguration on host1 until it becomes available again. The configuration for each database is shown in Table 14.

*Table 14. Configuration values for each HADR database after a failover. Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured*

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
<b>hadr_target_list</b>	host2:40   host3:41   host4:42	host1:10   host3:41   host4:42	host2:40   host1:10   host4:42	host2:40   host1:10   host3:41
<b>hadr_remote_host</b>	host2	<b>host3</b>	<b>host2</b>	<b>host3</b>
<b>hadr_remote_svc</b>	40	<b>41</b>	<b>40</b>	<b>41</b>
<b>hadr_remote_inst</b>	dbinst2	<b>dbinst3</b>	<b>dbinst2</b>	<b>dbinst3</b>
<b>hadr_local_host</b>	host1	host2	host3	host4
<b>hadr_local_svc</b>	10	40	41	42
Configured <b>hadr_syncmode</b>	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective <b>hadr_syncmode</b>	n/a	<b>SUPERASYNC</b>	n/a	<b>SUPERASYNC</b>

After a short period of time, host1 becomes available. The DBA tries to start host1 as a standby, but because host1 has more logs than were propagated to host3, host1 is rejected as part of the initial handshake with the new primary. The DBA takes a backup of the new primary, restores it to host1, and starts HADR on that host:

```
DB2 BACKUP DB hadr_db
```

```
DB2 RESTORE DB hadr_db
```

```
DB2 START HADR ON DB hadr_db AS STANDBY
```

As is shown in Table 15, host1 is reconfigured.

*Table 15. Configuration values for a reintegrated standby. Various rows in column 2 have been bolded to show that they have been auto-reconfigured*

Configuration parameter	Host1	Host2	Host3	Host4
<b>hadr_target_list</b>	host2:40   host3:41   host4:42	host1:10   host3:41   host4:42	host2:40   host1:10   host4:42	host2:40   host1:10   host3:41
<b>hadr_remote_host</b>	<b>host3</b>	host3	host2	host3
<b>hadr_remote_svc</b>	<b>41</b>	41	40	41
<b>hadr_remote_inst</b>	<b>dbinst3</b>	dbinst3	dbinst2	dbinst3
<b>hadr_local_host</b>	host1	host2	host3	host4
<b>hadr_local_svc</b>	10	40	41	42
Configured <b>hadr_syncmode</b>	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective <b>hadr_syncmode</b>	<b>SUPERASYNC</b>	SUPERASYNC	n/a	SUPERASYNC

If the DBA wants to make host1 the primary again, then all that is required is a failback, which will restore the original configuration shown in Table 12 on page 214.

## An auxiliary standby takes over by force (failover) in a SA MP environment

This example is similar to the previous one, but HADR has been deployed with IBM Tivoli System Automation for Multiplatforms (SA MP) to automate failover.

A power failure in City A results in the principal standby (host2) becoming unavailable. Following that, there is an outage on the primary (host1). Normally, SA MP, the cluster manager, would automatically fail over to the principal standby (host2), but the power outage means that one of the auxiliary standbys needs to be the takeover target. Failover cannot be automated to auxiliary standbys, so the DBA must do it manually. However, before doing this, the DBA needs to ensure that SA MP is disabled so that if host1 or host2 become available, there is no possibility for a *split brain* situation, in which more than one database is operating independently as a primary. To do this, the DBA issues the following command on host1 and host2 (whenever they become available):

```
db2haicu -disable
```

In addition, the DBA needs to keep host1 offline to eliminate the possibility that the old primary will restart if a client connects to it.

The DBA queries the two auxiliary standbys to determine which one has the most log data:

```
db2pd -hadr -db hadr_db | grep 'STANDBY_LOG_FILE,PAGE,POS'
```

The DBA determines that host3 is the most up to date and picks that host as the new primary.

Then, the DBA issues the force takeover on host3:

```
DB2 TAKEOVER HADR ON DB hadr_db BY FORCE
```

After the takeover is completed successfully, host3 becomes the new primary. Meanwhile, host2 becomes available again. host3 informs host2 and host4 that it is now the primary. On host3, the values for **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** are reconfigured to point to host2, which is the principal standby because it is the first entry in the **hadr\_target\_list** on host3. On host2, the synchronization mode is reconfigured to SUPERASYNC because that is the setting for **hadr\_syncmode** on host3; in addition, the **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** are updated (persistently). host4 is automatically redirected to the new primary, host3. In this redirection, host4 updates (persistently) its **hadr\_remote\_host**, **hadr\_remote\_svc**, and **hadr\_remote\_inst** configuration parameters. There is no automatic reconfiguration on host1. The configuration for each database is shown in Table 16.

*Table 16. Configuration values for each HADR database after a failover. Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured*

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
<b>hadr_target_list</b>	host2:40   host3:41   host4:42	host1:10   host3:41   host4:42	host2:40   host1:10   host4:42	host2:40   host1:10   host3:41
<b>hadr_remote_host</b>	host2	<b>host3</b>	<b>host2</b>	<b>host3</b>
<b>hadr_remote_svc</b>	40	<b>41</b>	<b>40</b>	<b>41</b>
<b>hadr_remote_inst</b>	dbinst2	<b>dbinst3</b>	<b>dbinst2</b>	<b>dbinst3</b>
<b>hadr_local_host</b>	host1	host2	host3	host4
<b>hadr_local_svc</b>	10	40	41	42
Configured <b>hadr_syncmode</b>	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective <b>hadr_syncmode</b>	n/a	<b>SUPERASYNC</b>	n/a	<b>SUPERASYNC</b>

## High availability disaster recovery (HADR) in Db2 pureScale environments

Db2 high availability disaster recovery (HADR) is supported in a Db2 pureScale environment that provides excellent continuous availability. With HADR, you also have DR (disaster recovery) protection. With a second copy of data on the standby site, you are protected from total failure at the primary site.

Configuring and managing HADR in a Db2 pureScale environment is very similar to configuring and managing HADR in other environments. You create a standby

database by restoring using a backup image or split mirror from the primary database, set various HADR configuration parameters, and start HADR on the standby and then the primary. The standby can quickly take over as the primary in the event of a role switch or a failover. All the administration commands are the same as what you are used to with HADR in other environments, but you can use only the **db2pd** command and the MON\_GET\_HADR table function to monitor HADR. Other monitor interfaces such as snapshot **do not** report HADR information in Db2 pureScale environments.

There are, however, some important differences for HADR in Db2 pureScale environments. An HADR pair is made up of a primary cluster and a standby cluster. Each cluster is made up of multiple members and at least one cluster caching facility; the member topology must be the same in the two clusters. The member from which you issue the **START HADR** command, on both the primary and the standby, is designated as the *preferred replay member*. When the database operates as a standby, only one member (the replay member) is activated. The database selects the preferred replay member as the replay member if the Db2 instance is online on the member, otherwise, another member is selected. That replay member replays all of the logs, and the other members are not activated. An HADR TCP connection is established between each member on the primary and the current replay member on the standby. Each member on the primary ships its logs to the standby replay member through the TCP connection. The HADR standby merges and replays the log streams. If the standby cannot connect to a particular member, A, on the primary (because of network problems or because the member is inactive) another member, B, on the primary that can connect to the standby sends the logs for member A to the standby. This process is known as *assisted remote catchup*.

## Restrictions for HADR in Db2 pureScale environments

There are a number of restrictions that you should be aware of if you are planning to deploy HADR in a Db2 pureScale environment.

The restrictions are as follows:

- A peer window is not supported. The **hadr\_peer\_window** configuration parameter must be set to 0.
- You cannot have more than one HADR standby database.
- The topology of the primary and the standby must be synchronized. If you add a member on the primary, that operation is replayed on the standby. If you drop a member on the primary, you must reinitialize the standby by using a backup or a split mirror from the primary's new topology.
- The reads on standby feature is not supported.
- You cannot use the integrated cluster manager, IBM Tivoli System Automation for Multiplatforms (SA MP), to manage automated failover; it manages high availability within the local cluster only.
- Network address translation (NAT) between the primary and standby sites is not supported.

## HADR setup in a Db2 pureScale environment

There are a few considerations for setting up an HADR database pair in a Db2 pureScale environment.

### Asymmetric standby members

Because only one member on the standby replays logs, consider configuring a standby member with more CPU power and memory to serve as the preferred

replay member. Similarly, consider which member on the primary cluster has the most CPU and memory, so that you can select it to be the preferred replay member if the current primary become the standby after a role switch. In both cases, you designate the preferred replay member by issuing the **START HADR** command from that member, with either AS PRIMARY or AS STANDBY option. The preferred replay member designation is persistent. It remains in place until changed by the next **START HADR** command.

Because the member topologies on the primary and standby must be the same, if you add members on the primary, you must also add members on the standby. If you have resource constraints, such as hardware constraints, you can configure the new standby members as logical members that share hosts. If the standby takes over the primary role, this new primary will not be as powerful as the old primary.

### Standby cluster caching facilities

The cluster caching facility (CF) does not have to be the same on the primary and standby clusters. The standby makes minimal use of the CF because only one member performs the replay, so it is possible to have only one CF on the standby cluster. If, however, the standby takes over as the new primary, you should add a CF to help ensure that your Db2 pureScale environment is highly available. Adding that secondary CF requires an outage because you cannot add it without stopping the Db2 pureScale instance.

### Member subsetting

You can use member subsetting to specify a subset of members to use for a database. The subset is stored internally as a list of member numbers. The database then maps the members to host names for the client to connect to. If this database uses HADR, you can only specify the subset list on the primary database. The subset member list is replicated to the standby.

### High availability disaster recovery (HADR) monitoring in a Db2 pureScale environment

You must use the **db2pd** command or the MON\_GET\_HADR table function to monitor your HADR databases in a Db2 pureScale environment.

Other interfaces, such as the **GET SNAPSHOT FOR DATABASE** command, the SNAPHADR administrative view, and the SNAP\_GET\_HADR table function, do not return any HADR information, so it will appear as if HADR is not configured.

The **db2pd** command and the MON\_GET\_HADR table function return essentially the same information, but because reads on standby is not supported in a Db2 pureScale environment, you can only use the **db2pd** command to monitor from a standby. As well, the **db2pd** command is preferred during takeover because there could be a time window during which neither the primary nor the standby allows client connections.

### db2pd command

In a Db2 pureScale environment, the **db2pd** command returns a section in its output for each log stream being processed by the member on which the command is issued. On the primary, any assisted remote catchup streams are reported by their owner (that is, the assisted member) and the assisting member. On the standby, all of the streams are returned if the command is issued on the replay member; if you issue **db2pd** on a non-replay member, no data is returned.



If you want to see all of the log streams from the primary, use the `-allmembers` and `-hadr` options with the **db2pd** command. If you use the `-allmembers` option on the standby, for each non replay member, the output indicates that database is not active on the member; for the replay member, all streams are returned. As a result, this command option is only useful on the standby if you want to find out which member is the current replay member (alternatively, you can check the `STANDBY_MEMBER` field in the **db2pd** output from the primary).

The following example is for an HADR setup using three-member clusters: members 0, 1, and 2. Member 1 is active but it is in assisted remote catchup state and is being assisted by member 0; the standby replay member is member 0. The DBA issues the **db2pd** command for member 0 on the primary. Notice that two sets of data are returned: one for each log stream member 0 is processing:

```
Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23
```

```

      HADR_ROLE = PRIMARY
      REPLAY_TYPE = PHYSICAL
      HADR_SYNCMODE = ASYNC
      STANDBY_ID = 1
      LOG_STREAM_ID = 0
      HADR_STATE = PEER
      HADR_FLAGS =
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS.ibm.com
STANDBY_INSTANCE = db2inst
STANDBY_MEMBER = 0
      HADR_CONNECT_STATUS = CONNECTED
      HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
      HEARTBEAT_INTERVAL(seconds) = 25
      HADR_TIMEOUT(seconds) = 100
      TIME_SINCE_LAST_RECV(seconds) = 3
      PEER_WAIT_LIMIT(seconds) = 0
      LOG_HADR_WAIT_CUR(seconds) = 0.000
      LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
      LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
      LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
      PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
      STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
      HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
      PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
      STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
      STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 13000
STANDBY_SPOOL_PERCENT = 0
      PEER_WINDOW(seconds) = 0
      READS_ON_STANDBY_ENABLED = N

      HADR_ROLE = PRIMARY
      REPLAY_TYPE = PHYSICAL
      HADR_SYNCMODE = ASYNC
      STANDBY_ID = 1
      LOG_STREAM_ID = 1
      HADR_STATE = REMOTE_CATCHUP
      HADR_FLAGS = ASSISTED_REMOTE_CATCHUP ASSISTED_MEMBER_ACTIVE
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst
PRIMARY_MEMBER = 0

```

```

STANDBY_MEMBER_HOST = hostS.ibm.com
STANDBY_INSTANCE = db2inst
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
HEARTBEAT_INTERVAL(seconds) = 25
HADR_TIMEOUT(seconds) = 100
TIME_SINCE_LAST_RECV(seconds) = 16
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.005631
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.837
LOG_HADR_WAIT_COUNT = 124
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
PRIMARY_LOG_FILE,PAGE,POS = S0000012.LOG, 1, 56541576
STANDBY_LOG_FILE,PAGE,POS = S0000012.LOG, 1, 56541576
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000012.LOG, 1, 56541576
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:25.000000 (1307566165)
STANDBY_LOG_TIME = 06/08/2011 13:49:25.000000 (1307566165)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:25.000000 (1307566165)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 13000
STANDBY_SPOOL_PERCENT = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = N

```

## MON\_GET\_HADR table function

In a Db2 pureScale environment, the MON\_GET\_HADR table function returns a row for each log stream. The table function cannot be issued on the standby because reads on standby is not supported in a Db2 pureScale environment. Use the LOG\_STREAM\_ID field in the table function output to identify the log stream and the PRIMARY\_MEMBER and STANDBY\_MEMBER fields to identify the members processing the stream on the primary and standby sides.

The table function takes a member argument and returns the stream that the specified member owns and all remote catchup streams that it is assisting. If the argument is an assisting member, the assisted remote catchup streams have their HADR\_STATE field reported as being in REMOTE\_CATCHUP with the ASSISTED\_REMOTE\_CATCHUP flag set in the HADR\_FLAGS field. If the argument is an assisted member, the assisted remote catchup stream has its HADR\_STATE field reported as DISCONNECTED.

If you specify -1 or NULL as the argument, the results for the current database member (that is, the member processing the query) are returned. If you specify -2 as the argument, the results for all members on the primary are returned. Any assisted remote catchup streams are reported on the assisting member only. If a member is inactive and assisted remote catchup has not yet been established for that member's log stream, that log stream does not appear in the output. The table function request is passed to all active members and the results are merged, so inactive members are not represented.

In the following examples, the DBA calls the MON\_GET\_HADR table function for monitoring an HADR setup using three-member clusters: members 0, 1, and 2. Member 1 is active but it is in assisted remote catchup state and is being assisted by member 0; the standby replay member is member 0. The DBA calls the table

function with argument 0, 1, 2 and -2 (for all members). Notice that two rows are returned when the argument is 0: one for each log stream that member 0 is processing:

#### Example for member 0

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE, HADR_FLAGS
from table (mon_get_hadr(0))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE	HADR_FLAGS
0	0	0	PEER	
1	0	0	REMOTE_CATCHUP	ASSISTED_REMOTE_CATCHUP

#### Example for member 1

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE, HADR_FLAGS
from table (mon_get_hadr(1))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE	HADR_FLAGS
1	1	0	DISCONNECTED	

#### Example for member 2

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE, HADR_FLAGS
from table (mon_get_hadr(2))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE	HADR_FLAGS
2	2	0	PEER	

#### Example for all members

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE, HADR_FLAGS
from table (mon_get_hadr(-2))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE	HADR_FLAGS
0	0	0	PEER	
1	0	0	REMOTE_CATCHUP	ASSISTED_REMOTE_CATCHUP
2	2	0	PEER	

## HADR standby replay in a Db2 pureScale environment

In a Db2 pureScale environment, only one member of the HADR standby cluster replays logs, and other members remain inactive. Members in the HADR primary cluster ship their logs to the replay member directly by using a TCP connection or indirectly through assisted remote catchup.

When a standby database is started, standby replay service is activated on the member that is designated as the *preferred replay member* if the Db2 instance is online on the member. Otherwise, replay is activated on another member. There is no way to control the selection among non preferred members; however, any member that is running in restart light mode (that is, a member that is not active on its home host) is given the lowest priority. Even though the preferred replay member designation is persistent, if replay is active on another member because the preferred replay member was not available, replay does not automatically revert to the preferred member when that member becomes available. The only way to force replay onto the preferred replay member is to deactivate and reactivate HADR on the standby.

Because the replay member on the standby replays logs that are generated by all members on the primary, there is a possibility that it can become a bottleneck. To avoid a potential impact, you should select the member with more resources, such as CPU and memory, as the preferred replay member. You implicitly designate the preferred replay member by issuing the **START HADR** command on it. The member from which you issue the **START HADR AS STANDBY** command is the preferred replay member on the standby cluster; the member from which you issue the **START HADR**

**AS PRIMARY** command is the preferred replay member on the primary cluster. The status of preferred replay member on the primary takes effect only when the primary becomes a standby

If the current replay member goes down abnormally (for example, as a result of a software or hardware failure) or normally (for example, as a result of a user command to deactivate the particular member), replay is automatically migrated to another member. If the current replay member goes down abnormally, member crash recovery occurs, and a member is selected to resume replay, with preference to the preferred replay member during the selection (the old replay member might or might not be reselected). As long as there is one online member in the standby cluster, replay continues. To stop replay, deactivate the whole standby database.

You can find out which member is the current replay member from the primary or the standby. On the primary, use the **db2pd** command with the **-hadr** parameter or the **MON\_GET\_HADR** table function. The replay member is indicated in the **STANDBY\_MEMBER** field. If you want to determine the current replay member from the standby, you can use only the **db2pd** command because the table function cannot be called from a standby in a Db2 pureScale environment. Because you do not know which replay member is active, you must issue the following command:

```
db2pd -hadr -db DB_name -allmembers
```

In the output, only the current replay member has HADR information; all non-replay members show Database *DB\_name* not activated on database member *X*.

### Changing the preferred replay member:

You designate a preferred replay member by issuing the **START HADR** command on that member. If you want to change that designation, you have to reissue the command.

Note that if a database is already active and in the desired role, the **START HADR** command will be a nop (no operation performed) that returns an error, and the preferred replay member is not updated. Use the following procedure to designate or redesignate the preferred replay member.

### About this task

The preferred replay member is the member that is preferred for replaying logs on an HADR standby database in a Db2 pureScale environment. On a standby, it still might not be the member doing the actual replay. On the primary, it is the first member that the standby replay service attempts to start on if that primary becomes the standby. The preferred replay member designation is persistent and can only be changed by starting and stopping HADR.

### Procedure

To designate a preferred replay member

- On the standby:
  1. Issue the **DEACTIVATE DATABASE** command from any member in the standby cluster.
  2. Issue the **START HADR** command with the **AS STANDBY** option on the member that you want to designate as the preferred replay member.
- On the primary:

1. Issue the **STOP HADR** command from any member in the primary cluster.

**Note:** The primary remains active during this procedure.

2. Issue the **START HADR** command with the AS PRIMARY option on the member that you want to designate as the preferred replay member.

**Note:** This designation only takes effect if the primary database switches to the standby role.

## Results

If the **START HADR** command completes successfully, the preferred replay member is updated. If the **START HADR** command fails, the preferred member might or might not have been updated, depending on how far the command execution went. To ensure that the designation applies to the correct member, rerun the procedure described in this task.

## Db2 pureScale topology changes and high availability disaster recovery (HADR)

In a Db2 pureScale environment, making changes to the HADR primary cluster and HADR standby cluster can require an outage.

In general, the primary and standby clusters must have the same member topology; that is, each instance must have the same number of members with the same member IDs. The only exception is when you add members to the standby. You can add members when the database is either offline or online. If you drop a member from the primary cluster (dropping a member is not allowed on the standby), you must stop HADR, deactivate the primary, and reinitialize the standby.

If you add a cluster caching facility (CF), you also require an outage on the Db2 pureScale instance.

### Adding members to a high availability disaster recovery (HADR) setup:

You can scale out your Db2 pureScale instance by adding members without impacting your HADR setup. You can add members online or offline.

## Procedure

To add a member to an HADR setup in a Db2 pureScale instance:

1. Add the new member to the standby cluster. From a standby host that is part of the Db2 pureScale instance, issue the following command:

```
db2iupdt -add -m memberHostName-mnet MemberNetName -mid MemberID InstName
```

This command adds the member to the member topology but not to the database topology.

2. Update the member-specific configuration parameters for the new member on the standby:

```
UPDATE DATABASE CONFIGURATION FOR db_name MEMBER member_ID  
USING hadr_local_host standby_member_host
```

```
UPDATE DATABASE CONFIGURATION FOR db_name MEMBER member_ID  
USING hadr_local_svc standby_member_port
```

3. Add the new member to the primary cluster. From a primary host that is part of the Db2 pureScale instance, issue the following command:

```
db2iupdt -add -m memberHostName-mnet MemberNetName -mid MemberID InstName
```

You must use the same member ID that you specified when adding the member to the standby cluster. This command adds the member to the member topology but not to the database topology.

4. Update the member-specific configuration parameters for the new member on the primary:

```
UPDATE DATABASE CONFIGURATION FOR db_name MEMBER member_ID
  USING hadr_local_host primary_member_host
UPDATE DATABASE CONFIGURATION FOR db_name MEMBER member_ID
  USING hadr_local_svc primary_member_port
```

On the primary this member is not listed because it is not currently activated. On the standby this member is not listed because it does not yet exist in the database topology.

5. Activate the new member on the primary by doing one of the following steps:
  - Connect to the database on the new member.
  - Issue the **ACTIVATE DATABASE** command.

If you have not added the member to the standby cluster by the time that it receives the add member log record that results from the member activation on the primary, the standby database will be shut down.

### What to do next

Add the new member to the target list on the primary and the standby.

### Removing members from a high availability disaster recovery (HADR) setup:

Removing a member from your Db2 pureScale instance requires you to reinitialize the standby based on the primary's updated topology.

### About this task

To drop a member, you need to stop HADR and the Db2 pureScale instance. You cannot drop the last member in the instance using this procedure.

### Procedure

To remove a member from an HADR setup in a Db2 pureScale instance:

1. Remove the member from the primary cluster. You must do this from a host that will still belong to the instance after the member is dropped.
  - a. Stop HADR on the primary database using the **STOP HADR** command.
  - b. Stop the Db2 pureScale instance using the **db2stop** command.
  - c. Drop the member by running the following command:

```
db2iupdt -drop -m member_ID instance_name
```

**Note:** You cannot directly drop a member from an HADR standby database.

2. Remove the member from the standby cluster. You must do this from a host that will still belong to the instance after the member is dropped.
  - a. Deactivate the database on the standby database using the **DEACTIVATE DATABASE** command.

```
DEACTIVATE DATABASE db_name
```
  - b. Drop the database using the following command:

```
DROP DATABASE db_name
```

- c. Drop the member by running the following command:

```
db2iupdt -drop -m member_ID instance_name
```

You must use the same member ID that you specified when removing the member from the primary cluster.

3. Create the standby database by restoring a backup image or by initializing a split mirror, based on the primary's updated topology after step 1.
  - a. On the primary, issue the following command:  

```
BACKUP DB dbname
```
  - b. Restore the standby by issuing the following command:  

```
RESTORE DB dbname
```
4. Update the HADR-specific database configuration parameters on the standby cluster.
5. Start HADR on the primary:  

```
START HADR AS PRIMARY
```
6. Start HADR on the standby:  

```
START HADR AS STANDBY
```

## HADR takeover operations in a Db2 pureScale environment

When an HADR standby database takes over as the primary database in a Db2 pureScale environment, there are a number of important differences from HADR in other environments.

With HADR, there are two types of takeover: *role switch* and *failover*. Role switch, sometimes called graceful takeover or non-forced takeover, can be performed only when the primary is available and it switches the role of primary and standby. Failover, or forced takeover, can be performed when the primary is not available. It is commonly used in primary failure cases to make the standby the new primary. The old primary remains in the primary role in a forced takeover, but the standby sends it a message to disable it. Both types of takeover are supported in a Db2 pureScale environment, and both can be issued from any of the standby database members and not just the current replay member. However, after the standby completes the transition to the primary role, the database is only started on the member that served as the replay member before the takeover. The database can be started on the other members by issuing an **ACTIVATE DATABASE** command or implicitly through a client connection.

### Role switch

After a role switch, which is initiated by issuing the **TAKEOVER HADR** command from any standby member, the standby cluster becomes the primary cluster and vice versa. Role switch helps ensure that no data is lost between the old primary and new primary. You can initiate a role switch in the following circumstances only:

- Crash recovery is not occurring on the primary cluster, including member crash recovery that is pending or in progress.
- All the log streams are in peer or assisted remote catchup state.
- All the log streams are in remote catchup state or in assisted remote catchup state, and the synchronization mode is SUPERASYNC.

Before you initiate a role switch in remote catchup or assisted remote catchup state, check the log gap between the primary and standby log streams. A large gap can result in a long takeover time because all of the logs in that gap must be shipped and replayed first.

During a role switch, the following steps occur on the primary:

1. New connections are rejected on all members, any open transactions are rolled back, and all remaining logs are shipped to the standby.
2. The primary cluster's database role changes to standby.
3. A member that has a direct connection to the standby is chosen as the replay member, with preference given to the preferred replay member (that is, the member that HADR was started from).
4. Log receiving and replay starts on the replay member.
5. The database is shut down on the other non-replay members of the cluster.

And the following steps occur on the standby:

1. Log receiving is stopped on the replay member after the end of logs is reached on each log stream, helping ensure no data loss.
2. The replay member finishes replaying all received logs.
3. After it is confirmed that the primary cluster is now in the standby role, the replay member changes the standby cluster's role to primary.
4. The database is opened for client connections, but it is only activated on the member that was previously the standby replay member.

## Failover

After a failover, which is initiated by issuing the **TAKEOVER HADR** command with the **BY FORCE** option from any standby member, the standby cluster becomes the primary cluster. The old primary cluster is sent a disabling message, but its role is not changed. Any member on the primary that receives this message disables the whole primary cluster. By initiating a failover, you are accepting the trade-off between potential data loss and having a working database. You cannot initiate a failover if the databases are in local catchup state.

**Note:** Unlike in previous releases, you can now initiate a failover even if log archive retrieval is in progress.

During a failover, the following steps occur on the primary (assuming it is online and connected to the standby):

1. After it receives the disabling message, the database is shut down and log writing is stopped.

And the following steps occur on the standby, all of which are carried out from the replay member:

1. A disabling message is sent to the primary, if it is connected.
2. Log shipping and log retrieval is stopped, which entails a risk of data loss.
3. The replay member finishes replaying all received logs (that is, the logs that are stored in the log path).
4. Any open transactions are rolled back.
5. The replay member changes the standby cluster's role to primary.
6. The database is opened for client connections, but it is only activated on the member that was previously the standby replay member.



You can reintegrate the old primary as a new standby only if its log streams did not diverge from the new primary's log streams. Before you can start HADR, the database must be offline on all of the old primary's members; the cluster caching facilities, however, can stay online. If any members are online, kill them instead of issuing the **DEACTIVATE DATABASE** command on them.

## Scenario: Deploying HADR in a Db2 pureScale environment

This scenario describes the planning, configuring, and deploying of a high availability disaster recovery (HADR) setup for an online travel service called ExampleFlightsExpress (EFE), which is currently using the Db2 pureScale Feature. All these steps can be done without any downtime.

### Background

EFE chose to use the Db2 pureScale Feature for two reasons:

- Continuous availability. Downtime is fatal in the online retailing business, where customers are literally accessing services 24x7.
- Scalability. Because customer demand ebbs and flows depending on the time of year, EFE must be able to add capacity easily and without taking an outage.

EFE is already configured with “Archive logging” on page 24. EFE's setup is resilient unless there is a widespread outage that brings down the whole Db2 pureScale cluster. To address this shortcoming, EFE is going to use HADR, which is supported by the Db2 pureScale Feature. In a Db2 pureScale environment, HADR has a few limitations, such as no support for reads on standby, but those limitations are acceptable because EFE wants HADR only for disaster recovery.

### Planning

EFE is going to use the company's head office (site A) as the location for the HADR primary cluster and a regional office (site B), which is 500 miles (800 km) away as the location for the standby cluster. The two sites are connected by a WAN. Other details about the environment are as follows:

- Database name: `hadr_db`
- Instance owner on all hosts: `db2inst`
- TCP port that is used for HADR primary-standby communication: 4000
- TCP port that is used for SQL client/server communication: 8000
- Hosts for cluster caching facilities (with IDs 128 and 129) and members (with IDs 0, 1, 2, and 3) on the primary: `cfp0`, `cfp1`, `p0`, `p1`, `p2`, and `p3`
- Hosts for cluster caching facilities and members on the standby: `cfs0`, `cfs1`, `s0`, `s1`, `s2`, and `s3`

### Preferred replay members

Only one member on the standby performs all the replay for the logs that are generated on all the members on the primary. Therefore, EFE's DBA determines which member hosts in the primary and standby clusters have the most memory and CPU resources and designates those members as the preferred replay members. It is necessary to do this planning even for the primary because that designated member performs all the replay if the primary database fails and is reintegrated as the standby. On the primary, this is `p0` and on the standby, it is `s0`; in both cases, member 0 is the resident member on those hosts.

### Synchronization mode

EFE's DBA must choose between `ASync` (the default), `Sync`, `NEARSync`, and `SUPERASync` for the synchronization mode. To do this,

the DBA analyzes the WAN and determines that network throughput is 300 Mbit/second and that the round-trip time is 80 ms. Next, the DBA measures the logging rate, which is 20 MB/second at the cluster level. The network throughput is sufficient to support the logging rate and allow peak logging to reach 37 MB/second, so ASYNC is a suitable mode. If the throughput were closer to the logging rate, SUPERASYNC would be a better choice because it would allow the primary to get far ahead of the standby during peak transaction time.

### Scaling considerations

Because EFE tends to add temporary members during peak times of the year, EFE must decide how to scale out the standby, because the member topology must be the same across the HADR pair. To avoid additional costs, EFE decides that when it deploys additional members on the primary, on the standby cluster, it uses multiple members on the host machines. This would likely result in a less powerful database if the standby must take over as the new primary, but the savings are worth this potential drawback.

### Configuring HADR

The DBA performs the following steps:

1. The DBA takes an online backup of the intended primary database, `hadr_db`:  
`db2 BACKUP DB hadr_db online TO backup_dir`
2. The DBA restores the backup onto the intended standby cluster by issuing the following command:  
`db2 RESTORE DB hadr_db FROM backup_dir`
3. On the primary, the DBA sets the cluster-level HADR parameters that specify the standby cluster and the synchronization mode. Particularly important is the **`hadr_target_list`** parameter, which lists the remote members. Only one remote member is required to be listed in the **`hadr_target_list`**. Db2 retrieves the other members' addresses after the initial contact with the listed member. However, providing multiple addresses prevents a single point of failure, that is, the clusters cannot connect to each other if the one and only listed member is down. The DBA issues the following command:

```
db2 "UPDATE DB CFG FOR hadr_db USING
    HADR_TARGET_LIST {s0:4000|s1:4000|s2:4000|s3:4000}
    HADR_REMOTE_HOST {s0:4000|s1:4000|s2:4000|s3:4000}
    HADR_REMOTE_INST db2inst
    HADR_SYNCMODE      async"
```

Because there is only one standby, the **`hadr_remote_host`** parameter specifies the same group of addresses as the **`hadr_target_list`** parameter.

4. The DBA sets the member-level HADR parameters on the primary, which identify the address and port for each member:
  - For member 0:  
`db2 "UPDATE DB CFG FOR hadr_db MEMBER 0 USING
 HADR_LOCAL_HOST p0
 HADR_LOCAL_SVC 4000"`
  - For member 1:  
`db2 "UPDATE DB CFG FOR hadr_db MEMBER 1 USING
 HADR_LOCAL_HOST p1
 HADR_LOCAL_SVC 4000"`
  - For member 2:

- ```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 2 USING
      HADR_LOCAL_HOST p2
      HADR_LOCAL_SVC 4000"
```
- For member 3:
 

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 3 USING
      HADR_LOCAL_HOST p3
      HADR_LOCAL_SVC 4000"
```
5. On the standby, the DBA sets the cluster-level HADR parameters that specify the primary cluster and the synchronization mode:
- ```
db2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST {p0:4000|p1:4000|p2:4000|p3:4000}
      HADR_REMOTE_HOST {p0:4000|p1:4000|p2:4000|p3:4000}
      HADR_REMOTE_INST db2inst
      HADR_SYNCMODE async"
```
6. The DBA sets the member-level HADR parameters on the standby, which identify the address and port for each member:
- For member 0:
 

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 0 USING
      HADR_LOCAL_HOST s0
      HADR_LOCAL_SVC 4000"
```
  - For member 1:
 

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 1 USING
      HADR_LOCAL_HOST s1
      HADR_LOCAL_SVC 4000"
```
  - For member 2:
 

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 2 USING
      HADR_LOCAL_HOST s2
      HADR_LOCAL_SVC 4000"
```
  - For member 3:
 

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 3 USING
      HADR_LOCAL_HOST s3
      HADR_LOCAL_SVC 4000"
```

## Starting HADR

As with other HADR environments, the standby database must be started first. Because the member that the **START HADR** command is issued from is designated the preferred replay member, the DBA issues the following commands:

- From member 0 on the standby:
 

```
db2 START HADR ON DB hadr_db AS STANDBY
```
- From member 0 on the primary:
 

```
db2 START HADR ON DB hadr_db AS PRIMARY
```

To determine that HADR is up and running, the DBA calls the `MON_GET_HADR` table function from the primary:

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE
from table (mon_get_hadr(-2))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE
0	0	0	PEER
1	1	0	PEER
2	2	0	PEER
3	3	0	PEER

The DBA confirms that standby member 0, the preferred replay member, is indeed the current replay member by looking at the STANDBY\_MEMBER field. Every log stream reports the same member on the standby because all the members on the primary are connected to that standby member.

## Role switch

The DBA has to perform a role switch; that is, the current standby will take over the primary role, and the current primary will take over the standby role. This will allow some maintenance which requires a shutdown of the cluster to be performed at site A. This procedure takes place during a time of low usage in order to minimize impact on applications currently connected to the database.

1. The DBA ensures that none of the members on the primary are in an abnormal state:

```
SELECT ID,
       varchar(STATE,21) AS STATE,
       varchar(HOME_HOST,10) AS HOME_HOST,
       varchar(CURRENT_HOST,10) AS CUR_HOST,
       ALERT
FROM SYSIBMADM.DB2_MEMBER
```

ID	STATE	HOME_HOST	CUR_HOST	ALERT
0	STARTED	p0	p0	NO
1	STARTED	p1	p1	NO
2	STARTED	p2	p2	NO
3	STARTED	p3	p3	NO

4 record(s) selected.

2. The DBA ensures that all of the log streams are in PEER state:

```
select LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE
from table (mon_get_hadr(-2))
```

LOG_STREAM_ID	PRIMARY_MEMBER	STANDBY_MEMBER	HADR_STATE
0	0	0	PEER
1	1	0	PEER
2	2	0	PEER
3	3	0	PEER

3. At site B, the DBA issues the **TAKEOVER HADR** command on member 0:

```
TAKEOVER HADR ON DB hadr_db
```

After the command completes, member 0 on the new standby (the preferred replay member) is chosen as the replay member and the database is shut down on the other members on the standby cluster. On the new primary, the database is only activated on member 0; other members are activated with a client connection or if the DBA explicitly issues the **ACTIVATE DATABASE** command on each of them. Automatic client reroute sends any new clients to site B.

4. At site A, the DBA deactivates the database on the standby (this keeps the database in its role as an HADR standby):

```
DEACTIVATE DATABASE hadr_db
```

5. At site A, the DBA stops Db2 on the standby:

```
db2stop
```

6. At site A, the DBA performs the required maintenance.

7. At site A, the DBA starts Db2 on the standby:

```
db2start
```

8. At site A, the DBA activates the database on the standby:

```
ACTIVATE DATABASE hadr_db
```

The database is activated as an HADR primary with one replay member.

9. To revert to the original setup, the DBA issues the **TAKEOVER HADR** command on member 0 at site A:

```
TAKEOVER HADR ON DB hadr_db
```

## Failover

The DBA has to perform a failover; that is, an unexpected outage at site A requires that the standby at site B take over the primary role. An important difference for HADR in a Db2 pureScale environment is that there is no support for using IBM Tivoli System Automation for Multiplatforms (SA MP) to automate the failover (it's already being used to ensure high availability in the Db2 pureScale cluster). At any rate, in this scenario the DBA wants to have manual control over this kind of response to an outage.

1. The DBA performs a forced takeover from the standby database at site B.

```
TAKEOVER HADR ON DB hadr_db BY FORCE
```

The standby sends a disabling message to shut down the primary. After stopping log shipping and retrieval, the standby completes the replay of any logs in its log path. Finally, the standby becomes the new primary.

2. The DBA issues the **db2pd** command on the new primary to ensure that it has taken over the primary role.

```
db2pd -hadr -db hadr_db
```

3. After addressing the cause of the outage and getting site A up and running again, the DBA attempts to reintegrate the old primary as a standby.

```
START HADR ON DB hadr_db AS STANDBY
```

4. The DBA verifies that the site A is now the standby by checking the **HADR\_CONNECT\_STATUS** and **HADR\_STATE** fields to ensure that the show the database is connected and in either peer or remote catchup state.

```
db2pd -hadr -db hadr_db
```

Unfortunately, the log streams of the databases at the two sites have diverged, so the database is showing as disconnected. The DBA looks at the **diag.log** file of one of the members on the old primary and sees a message indicating that the database on site A cannot be made consistent with the new primary database.

5. The DBA has to drop the database and reinitialize it as an HADR standby at site A.

- a. Drop the database:

```
DROP DATABASE DB hadr_db
```

- b. Take a backup of the database at site B.

```
BACKUP DATABASE DB hadr_db ONLINE TO backup_dir
```

- c. Restore the backup image to site A.

```
db2 RESTORE DB hadr_db FROM backup_dir
```

- d. Set the cluster-level and member-level configuration parameters on the database at site A.

```
db2 "UPDATE DB CFG FOR hadr_db USING  
    HADR_TARGET_LIST {s0:4000|s1:4000|s2:4000|s3:4000}  
    HADR_REMOTE_HOST {s0:4000|s1:4000|s2:4000|s3:4000}  
    HADR_REMOTE_INST db2inst  
    HADR_SYNCMODE     async"
```

- For member 0:  

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 0 USING
      HADR_LOCAL_HOST p0
      HADR_LOCAL_SVC 4000"
```
  - For member 1:  

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 1 USING
      HADR_LOCAL_HOST p1
      HADR_LOCAL_SVC 4000"
```
  - For member 2:  

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 2 USING
      HADR_LOCAL_HOST p2
      HADR_LOCAL_SVC 4000"
```
  - For member 3:  

```
db2 "UPDATE DB CFG FOR hadr_db MEMBER 3 USING
      HADR_LOCAL_HOST p3
      HADR_LOCAL_SVC 4000"
```
6. The DBA wants to designate member 0 as the preferred replay member and issues the **START HADR** command from member 0 on the site A:  

```
db2 START HADR ON DB hadr_db AS STANDBY
```
  7. The DBA verifies that the site A is now the standby by checking the `HADR_CONNECT_STATUS` and `HADR_STATE` fields to ensure that the show the database is connected and is catching up to the primary.  

```
db2pd -hadr -db hadr_db
```
  8. To revert to the original setup, the DBA can perform a role switch as described in the previous section.

## HADR reads on standby feature

You can use the reads on standby capability to run read-only operations on the standby database in your High Availability and Disaster Recovery (HADR) solution. Read operations running on a standby do not affect the standby's main role of replaying logs shipped from the primary database.

The reads on standby feature reduces the total cost of ownership of your HADR setup. This expanded role of the standby database allows you to utilize the standby in new ways, such as running some of the workload that would otherwise be running on your primary database. This, in turn frees up the primary for additional workloads.

Read and write clients continue to connect to the primary database; however read clients can also connect to the read-enabled standby, or *active standby*, as long as it is not in the local catchup state or the replay-only window. An active standby's main role is still to replay logs shipped from the primary. As a result, the data on the standby should be virtually identical to the data on the primary. In the event of a failover, any user connections to the standby will be terminated while the standby takes over as the new primary database.

All types of read queries, including scrollable and non-scrollable cursors, are supported on the standby. Read capability is supported in all four HADR synchronization modes (SYNC, NEARSYNC, ASYNC, and SUPERASYNC) and in all HADR states except local catchup.

## Enabling reads on standby

You can enable the reads on standby feature on your High Availability and Disaster Recovery (HADR) standby database using the **DB2\_HADR\_ROS** registry variable.

### Before you begin

It is recommended that database configuration parameter **logindexbuild** be set to ON. This will prevent a performance impact from query access plans avoiding any invalid indexes.

It is also recommended that you use a virtual IP when you have reads on standby enabled. Client reroute does not differentiate between writable databases (primary and standard databases) and read-only databases (standby databases). Configuring client reroute between the primary and standby might route applications to the database on which they are not intended to be run.

### About this task

You cannot use automatic client reroute (ACR) if you enable reads on standby.

### Procedure

1. Set the **DB2\_HADR\_ROS** registry variable to ON.
2. Set up and initialize the primary and standby databases for HADR. Refer to “Initializing high availability disaster recovery (HADR)” on page 110.

### Results

Your standby database is now considered an *active standby*, meaning that it will accept read-only workloads.

### What to do next

You can now utilize your standby database as you see fit, such as configuring some of your read-only workload to run on it.

To enable your applications to maintain access to your standby database, follow the steps described in the “Continuous access to Read on Standby databases using Virtual IP addresses” white paper.

### Data concurrency on the active standby database

Changes on the HADR primary database may not necessarily be reflected on the HADR active standby database. Uncommitted changes on the primary may not replicate to the standby until the primary flushes, or sends, its logs to disk.

Logs are only guaranteed to be flushed to disk-and, therefore sent to the standby-after they have been committed. Log flushes can also be triggered by undeterministic conditions such as a log buffer full situation. As a result, it is possible for uncommitted changes on the primary to remain in the primary's log buffer for a long time. Because the logger avoids flushing partial pages, this situation may particularly affect small uncommitted changes on the primary.

If your workload running on the standby requires the data to be virtually identical to the data on the primary, you should consider committing your transactions more frequently.

### Isolation level on the active standby database:

The only isolation level that is supported on an active standby database (an HADR standby database that is read enabled) is Uncommitted Read (UR). If the isolation level requested by an application, statement, or sub-statement is higher than UR, an error will be returned (SQL1773N Reason Code 1).

If you require an isolation level other than UR, consider using the HADR primary instead of the standby for this application. If you simply want to avoid receiving this message, set the **DB2\_STANDBY\_ISO** registry variable to UR. When **DB2\_STANDBY\_ISO** is set to UR, the isolation level will be silently coerced to UR. This setting takes precedence over all other isolation settings such as statement isolation and package isolation.

### Replay-only window on the active standby database:

When an HADR active standby database is replaying DDL log records or maintenance operations, the standby enters the *replay-only window*. When the standby is in the replay-only window, existing connections to the standby are terminated and new connections to the standby are blocked (SQL1224N).

New connections are allowed on the standby after the replay of all active DDL or maintenance operations has completed.

The only user connections that can remain active on a standby in the replay-only window are connections that are executing **DEACTIVATE DATABASE** or **TAKEOVER** commands. When applications are forced off at the outset of the replay-only window, an error is returned (SQL1224N). Depending on the number of readers connected to the active standby, there may be a slight delay before the DDL log records or maintenance operations are replayed on the standby.

There are a number of DDL statements and maintenance operations that, when run on the HADR primary, will trigger a replay-only window on the standby. The following lists are not exhaustive.

#### DDL statements

- CREATE, ALTER, or DROP TABLE (except DROP TABLE for DGTT)
- CREATE GLOBAL TEMP TABLE
- TRUNCATE TABLE
- RENAME TABLE
- RENAME TABLESPACE
- CREATE, DROP, or ALTER INDEX
- CREATE or DROP VIEW
- CREATE, ALTER, or DROP TABLESPACE
- CREATE, ALTER, or DROP BUFFER POOL
- CREATE, ALTER, or DROP FUNCTION
- CREATE, ALTER, or DROP PROCEDURE
- CREATE or DROP TRIGGER
- CREATE, ALTER, or DROP TYPE
- CREATE, ALTER, or DROP ALIAS
- CREATE or DROP SCHEMA
- CREATE, ALTER, or DROP METHOD



- CREATE, ALTER, or DROP MODULE
- CREATE, ALTER, or DROP NICKNAME
- CREATE, ALTER, or DROP SEQUENCE
- CREATE, ALTER, or DROP WRAPPER
- CREATE, ALTER, or DROP FUNCTION MAPPING
- CREATE or DROP INDEX EXTENSION
- CREATE or DROP INDEX FOR TEXT
- CREATE or DROP EVENT MONITOR
- CREATE, ALTER, or DROP SECURITY LABEL
- CREATE, ALTER, or DROP SECURITY LABEL COMPONENT
- CREATE, ALTER, or DROP SECURITY POLICY
- CREATE or DROP TRANSFORM
- CREATE, ALTER, or DROP TYPE MAPPING
- CREATE, ALTER, or DROP USER MAPPING
- CREATE or DROP VARIABLE
- CREATE, ALTER, or DROP WORKLOAD
- GRANT USAGE ON WORKLOAD
- REVOKE USAGE ON WORKLOAD
- CREATE, ALTER, or DROP SERVICE CLASS
- CREATE, ALTER, or DROP WORK CLASS SET
- CREATE, ALTER, or DROP WORK ACTION SET
- CREATE, ALTER, or DROP THRESHOLD
- CREATE, ALTER, or DROP HISTOGRAM TEMPLATE
- AUDIT
- CREATE, ALTER, or DROP AUDIT POLICY
- CREATE or DROP ROLE
- CREATE, ALTER, or DROP TRUSTED CONTEXT
- REFRESH TABLE
- SET INTEGRITY

#### **Maintenance operations**

- Classic, or offline, reorg
- Inplace, or online, reorg
- Index reorg (indexes all, individual index)
- MDC and ITC reclaim reorg
- Load
- Bind or rebind
- db2rbind
- Runstats
- Table move
- Auto statistics
- Auto reorg
- Real Time Statistics

#### **Other operations or actions**

- Automatic Dictionary Creation for tables with COMPRESS YES attribute

- Asynchronous Index Cleanup on detached table partition
- Implicit rebind
- Implicit index rebuild
- Manual update of statistics.
- Deferred MDC rollout
- Asynchronous Index cleanup after MDC rollout
- Reuse of a deleted MDC or ITC block on insert into MDC or ITC table
- Asynchronous background processes updating catalog tables SYSJOBS and SYSTASKS for inserting, updating, and deleting tasks

### Diagnostic log messages during replay-only window

When a replay-only window is active, the db2diag.log will contain several diagnostic messages, which indicate the start and end of the replay-only window. When the registry variable **DB2\_HADR\_REPLAY\_ONLY\_WINDOW\_DIAGLEVEL** is enabled for it, additional details about the DDL operation or utility invocation which caused the replay-only window are also displayed. Note that the **LOG\_DDL\_STMTS** database configuration parameter should be set to YES on the Primary database for DDL statement text to be transmitted to the Standby so that it can be displayed.

For example:

```
2017-11-22-15.48.36.321657-300 I522432E731          LEVEL: Info
PID      : 25476                      TID : 140255646705408 KTID : 26687<
PROC     : db2sysc
INSTANCE: db2inst1                    NODE : 000 DB   : TESTDB
APPHDL   : 0-8                        APPID: *LOCAL.DB2.171122204107
HOSTNAME: myhost1
EDUID    : 87
EDUNAME: db2redom (TESTDB)
FUNCTION: DB2 UDB, recovery manager, SQLP_REPLAY_ONLY_WINDOW_STAT::sqlpStartHadrReplayOnlyWindow, probe:9140
MESSAGE : Replay only window is triggered by this log record: LogStreamId / TID<
          / LSO / action
DATA #1 : SQLP_TID, PD_TYPE_SQLP_TID, 6 bytes
000000000FFA
DATA #2 : unsigned integer, 8 bytes
47025987
DATA #3 : db2LogStreamIDType, PD_TYPE_DB2_LOG_STREAM_ID, 2 bytes
0
DATA #4 : String, 3 bytes
DDL

2017-11-22-15.48.36.326071-300 E523164E517          LEVEL: Warning
PID      : 25476                      TID : 140255646705408 KTID : 26687<
PROC     : db2sysc
INSTANCE: db2inst1                    NODE : 000      DB   : TESTDB
APPHDL   : 0-8                        APPID: *LOCAL.DB2.171122204107
HOSTNAME: myhost1
EDUID    : 87                        EDUNAME: db2redom (TESTDB)
FUNCTION: DB2 UDB, base sys utilities, sqeLocalDatabase::HdrForceAppsInReplayOnlyWindow, probe:100
DATA #1 : String, 28 bytes
Replay only window is active

2017-11-22-15.48.36.334268-300 I524198E504          LEVEL: Info
PID      : 25476                      TID : 140255646705408 KTID : 26687<
PROC     : db2sysc
INSTANCE: db2inst1                    NODE : 000      DB   : TESTDB
APPHDL   : 0-8                        APPID: *LOCAL.DB2.171122204107
HOSTNAME: myhost1
EDUID    : 87                        EDUNAME: db2redom (TESTDB)
FUNCTION: DB2 UDB, recovery manager, SQLP_REPLAY_ONLY_WINDOW_STAT::sqlpSetDDLStmtForHadrReplayOnlyWin
MESSAGE : DDL statement text
DATA #1 : String, 23 bytes
drop table test_script1
```

```

2017-11-22-15.49.28.915844-300 E546560E554          LEVEL: Warning
PID      : 25476                                TID : 140255646705408 KTID : 26687<
PROC     : db2sysc
INSTANCE: db2inst1                             NODE : 000          DB   : TESTDB
APPHDL   : 0-8                                APPID: *LOCAL.DB2.171122204107
HOSTNAME: myhost1
EDUID    : 87                                EDUNAME: db2redom (TESTDB)
FUNCTION: DB2 UDB, base sys utilities, sqeLocalDatabase::HdrEndRep
layOnlyWindow, probe:210
DATA #1 : String, 73 bytes
Replay only window is inactive, connections to Active Standby are allowed

```

## Monitoring the replay-only window

You can monitor the replay-only window using the **db2pd** command with the **-hadr** option (on either the standby or the primary) or the **MON\_GET\_HADR** table function (from the primary). The standby's status, including information about the replay-only window, is sent to the primary on every heartbeat.

There are three pertinent elements to monitor:

- **STANDBY\_REPLAY\_ONLY\_WINDOW\_ACTIVE**, which indicates whether DDL or maintenance-operation replay is in progress on the standby. Normally, the value is N, but when the replay-only window is active, the value is Y.
- **STANDBY\_REPLAY\_ONLY\_WINDOW\_START**, which indicates the time at which the current replay-only window (if there is one) became active.
- **STANDBY\_REPLAY\_ONLY\_WINDOW\_TRAN\_COUNT**, which indicates the total number of existing uncommitted DDL or maintenance transactions executed so far in the current replay-only window (if there is one).

To use the table function, issue something similar to the following query on the primary:

```

select STANDBY_ID, STANDBY_REPLAY_ONLY_WINDOW_ACTIVE, STANDBY_REPLAY_ONLY_WINDOW_START,
STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT from table (mon_get_hadr(NULL))

```

Here is an example using the **db2pd** command on a standby that is currently in a replay-only window:

```

Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 -- Date 06/08/2011 13:57:23

```

```

      HADR_ROLE = STANDBY
      REPLAY_TYPE = PHYSICAL
      HADR_SYNCMODE = NEARSYNC
      STANDBY_ID = 1
      LOG_STREAM_ID = 0
      HADR_STATE = PEER
      HADR_FLAGS =
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS1.ibm.com
STANDBY_INSTANCE = db2inst
STANDBY_MEMBER = 0
      HADR_CONNECT_STATUS = CONNECTED
      HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
      HEARTBEAT_INTERVAL(seconds) = 25
      HADR_TIMEOUT(seconds) = 120
      TIME_SINCE_LAST_RECV(seconds) = 3
      PEER_WAIT_LIMIT(seconds) = 0
      LOG_HADR_WAIT_CUR(seconds) = 0.000
      LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
      LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
      LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
      PRIMARY_LOG_FILE,PAGE,POS = S00000009.LOG, 1, 49262315
      STANDBY_LOG_FILE,PAGE,POS = S00000009.LOG, 1, 49262315

```

```

HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
STANDBY_SPOOL_PERCENT = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = Y
STANDBY_REPLAY_ONLY_WINDOW_START = 06/08/2011 13:50:23
STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT = 5

```

## Recommendations for minimizing the impact of the replay-only window

Because replay operations on an HADR standby take priority over readers, frequent read-only windows can be disruptive to readers connected to or attempting to connect to the standby. To avoid or minimize this impact, consider the following recommendations:

- Run DDL and maintenance operations during a scheduled maintenance window, preferably at off-peak hours.
- Run DDL operations collectively rather than in multiple groups.
- Run **REORG** or **RUNSTATS** only on the required tables instead of all tables.
- Terminate applications on the active standby using the **FORCE APPLICATION** command with the **ALL** option before running the DDL or maintenance operations on the primary. Monitor the replay-only window to determine when it is inactive, and redeploy the applications on the standby.

## Temporarily terminating read applications on an active standby database

Although an HADR active standby database can be used to run read-only workloads, its main role is to replay log records in order to stay synchronized with the HADR primary in case it has to take over the primary's role.

In cases where the read-only workload is causing the standby to fall behind in its log replay, you might want to temporarily terminate all of the connections to the standby to allow it to catch up.

## About this task

Use the following procedure to temporarily make an active standby inaccessible to readers.

### Procedure

1. Issue the **FORCE APPLICATION** command. This terminates existing connections to the standby.
2. Change the virtual IP configuration. This prevents new connections to the standby.

## What to do next

After the standby has caught up with the primary through log replay, you need to revert the virtual IP configuration to its original setting to allow the connections to the active standby to resume.

## Reads on standby restrictions

You can use the reads on standby feature of high availability disaster recovery (HADR) to run read-only workloads on an HADR active standby database. In addition to the read-only restriction, this feature has other limitations that you should be aware of.

- The standby is inaccessible to user connections during the replay of DDL log records or maintenance operations (during the *replay-only window*).
- The standby is inaccessible to user connections when it is in the local catchup state.
- The reads on standby feature is not supported in Db2 pureScale environments.
- Only the uncommitted read (UR) isolation level is supported on the standby. Applications, statements, or sub-statements that request a higher isolation level receive an error.
- The instance-level audit configuration is not replicated to the standby. You must ensure that the instance-level auditing settings are the same on the primary and the standby by using the **db2audit** tool.
- Queries on the standby database can use only SMS system temporary table spaces.
- You cannot configure the standby as a federation server.
- The standby is inaccessible to user connections while it is actively replaying upgrade log records and is considered in an upgrade in progress state.

## Data and table types

- Declared global temporary tables (DGTs) are not supported on the standby.
- Created global temporary tables (CGTTs) can be created only on the primary database. Their definitions are replicated to the standby. However, access to CGTTs is not supported on the standby.
- The creation of CGTTs on the primary triggers the replay-only window on the standby.
- Tables with the NOT LOGGED INITIALLY (NLI) attribute cannot be accessed on the standby.
- Column-organized tables cannot be accessed on the standby.
- XML and large object (LOB) data must be inline to be successfully queried.
- You cannot query the following data: long field (LF), a distinct type based on LF data types, structured type columns, and varying-length string data (that is, data that resides in extended rows).

## Operations

- Write operations, namely operations that modify permanent database objects such as catalogs, tables, and indexes, are not permitted on the standby. In particular, you cannot perform any operation that generates log records on the standby.
- Explain tools (the **db2exfmt** and **db2expln** commands) and the **db2batch** command are not supported on the standby. If you want to analyze performance of the read-only workload, run these tools on the primary, make the necessary optimizations to the workload on the primary, and then move the optimized workload to the standby.
- Explicit binding of packages, explicit rebinding of packages, and implicit rebinding of packages are not supported on the standby. Attempts to run static packages that refer to invalidated objects result in an error. Instead,

bind the package on the primary and run the package on the standby after replicating the change to the standby.

- The self-tuning memory manager (STMM) is not supported on the standby. If you want to tune the standby, either to suit running the read-only workload or to help the standby to perform well after takeover, you must do so manually.
- Workload manager (WLM) DDL statements on the primary are replayed on the standby, but they are not effective on the standby. However, any definitions in the database backup that you used to set up the standby are active on a read-enabled standby.
- Creation and alteration of sequences is not supported on the standby. As well, you cannot use the NEXT VALUE expression to generate the next value in a sequence.
- Runtime revalidation of invalid objects is not supported on the standby.
- Backup and archive operations are not supported on the standby.
- Quiesce operations are not supported on the standby.
- The db2ReadLog API cannot be called on the standby.
- You cannot use the automatic client reroute (ACR) if you enable the reads on standby feature.

## Detecting and responding to system outages in a high availability solution

Implementing a high availability solution does not prevent hardware or software failures. However, having redundant systems and a failover mechanism enables your solution to detect and respond to failures, and reroute workload so that user applications are still able to do work.

### Procedure

When a failure occurs, your database solution must do the following:

1. Detect the failure.

Failover software can use heartbeat monitoring to confirm the availability of system components. A heartbeat monitor listens for regular communication from all the components of the system. If the heartbeat monitor stops hearing from a component, the heartbeat monitor signals to the system that the component has failed.

2. Respond to the failure: failover.

- a. Identify, bring online, and initialize a secondary component to take over operations for the failed component.
- b. Reroute workload to the secondary component.
- c. Remove the failed component from the system.

3. Recover from the failure.

If a primary database server fails, the first priority is to redirect clients to an alternate server or to failover to a standby database so that client applications can do their work with as little interruption as possible. Once that failover succeeds, you must repair whatever went wrong on the failed database server so that it can be reintegrated back into the solution. Repairing the failed database server may just mean restarting it.

4. Return to normal operations.

Once the failed database system is repaired, you must integrate it back into the database solution. You could reintegrate a failed primary database as the

standby database for the database that took over as the primary database when the failure occurred. You could also force the repaired database server to take over as the primary database server again.

## What to do next

Db2 database can perform some of these steps for you. For example:

- The Db2 High Availability Disaster Recovery (HADR) heartbeat monitor element, **hadr\_heartbeat**, can detect when a primary database has failed.
- Db2 client reroute can transfer workload from a failed database server to a secondary one.
- The Db2 fault monitor can restart a database instance that terminates unexpectedly.

## Administration notification log

The administration notification log (*instance\_name.nfy*) is the repository from which information about numerous database administration and maintenance activities can be obtained. A database administrator can use this information to diagnose problems, tune the database, or simply monitor the database.

The Db2 database manager writes the following kinds of information to the administration notification log on UNIX and Linux operating system platforms (on Windows operating system platforms, the event log is used to record administration notification events):

- Status of Db2 utilities such **REORG** and **BACKUP**
- Client application errors
- Service class changes
- Licensing activity
- File paths
- Storage problems
- Monitoring activities
- Indexing activities
- Table space problems

Administration notification log messages are also logged to the **db2diag** log files using a standardized message format.

Notification messages provide additional information to supplement the SQLCODE that is provided.

The administration notification log file can exist in two different forms:

### Single administration notification log file

One active administration notification log file, named *instance\_name.nfy*, that grows in size indefinitely. This is the default form and it exists whenever the **diagsize** database manager configuration parameter has the value of 0 (the default value for this parameter is 0).

### Rotating administration notification log files

A single active log file (named *instance\_name.N.nfy*, where *N* is the file name index that is a continuously growing number starting from 0), although a series of administration notification log files can be found in the location defined by the **diagpath** configuration parameter, each growing until reaching a limited size, at which time the log file is closed and a new

one is created and opened for logging with an incremented file name index (*instance\_name.N+1.nfy*). It exists whenever the **diagsize** database manager configuration parameter has a nonzero value.

**Note:** Neither single nor rotating administration notification log files are available on the Windows operating system platform.

You can choose which of these two forms exist on your system by appropriately setting the **diagsize** database manager configuration parameter.

## Configuration

The administration notification log files can be configured in size, location, and the types of events and level of detail recorded, by setting the following database manager configuration parameters:

### **diagsize**

The value of **diagsize** decides what form of administration notification log file will be adopted. If the value is 0, a single administration notification log file will be adopted. If the value is not 0, rotating administration notification log files will be adopted, and this nonzero value also specifies the total size of all rotating diagnostic log files and all rotating administration notification log files. The instance must be restarted for the new value of the **diagsize** parameter to take effect. See the “diagsize - Diagnostic log file size configuration parameter” topic for complete details.

### **diagpath**

Diagnostic information can be specified to be written to administration notification log files in the location defined by the **diagpath** configuration parameter. See the “diagpath - Diagnostic data directory path configuration parameter” topic for complete details.

### **notifylevel**

The types of events and the level of detail written to the administration notification log files can be specified with the **notifylevel** configuration parameter. See the “notifylevel - Notify level configuration parameter” topic for complete details.

**Note:** If the **diagsize** configuration parameter is set to a non-zero value, that value specifies the total size of the combination of all rotating administration notification log files and all rotating diagnostic log files contained within the diagnostic data directory. For example, if a system with 4 database partitions has **diagsize** set to 1 GB, the maximum total size of the combined notification and diagnostic logs can reach is 4 GB (4 x 1 GB).

## Detecting an unplanned outage

Before you can respond to the failure of a component, you must detect that the component failed. Db2 Data Server has several tools for monitoring the health of a database, or otherwise detecting that a database has failed.

You can configure these tools to notify you or take predefined actions when they detect a failure.

## Procedure

You can use the following tools to detect when a failure has occurred in some part of your Db2 database solution:



## Db2 fault monitor facility

The Db2 fault monitor facility keeps Db2 database instances up and running. When the Db2 database instance to which a Db2 fault monitor is attached terminates unexpectedly, the Db2 fault monitor restarts the instance. If your database solution is implemented in a cluster, you should configure the cluster managing software to restart failed database instances instead of the Db2 fault monitor.

## Heartbeat monitoring in clustered environments

Cluster managing software uses heartbeat messages between the nodes of a cluster to monitor the health of the nodes. The cluster manager detects that a node has failed when the node stops responding or sending any messages.

## Monitoring Db2 High Availability Disaster Recovery (HADR) databases

The HADR feature has its own heartbeat monitor. The primary database and the standby database each expect heartbeat messages from the other at regular intervals.

## High availability disaster recovery (HADR) monitoring:

Monitoring is an integral part of setting up and maintaining your HADR setup. The Db2 monitoring interfaces provide a detailed picture of the configuration and health of your environment.

You can use a number of methods to monitor the status of your HADR databases. There are two preferred ways of monitoring HADR:

- The `db2pd` command
- The `MON_GET_HADR` table function

In a Db2 pureScale environment, only these two interfaces return information about the HADR setup.

Alternatively, you can also use the following methods:

- The **GET SNAPSHOT FOR DATABASE** command
- The `db2GetSnapshot` API
- Other supported snapshot administrative views and table functions. There are some views and functions that are deprecated or discontinued.

## db2pd command

This command retrieves information from the Db2 memory sets. You can issue this command from either a primary database or a standby database. If you are using multiple standby databases and you issue this command from a standby, it does not return any information about the other standbys. If you issue this command from the primary, it returns information on all standbys.

To view information about high availability disaster recovery for database HADRDB, you could issue the following command:

```
db2pd -db HADRDB -hadr
```

Assuming you issued that command from the primary, you would receive something like the following sample output:

```
Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 --  
Date 06/08/2011 13:57:23
```

```

HADR_ROLE = PRIMARY
REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = SYNC
STANDBY_ID = 1
LOG_STREAM_ID = 0
HADR_STATE = PEER
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS1.ibm.com
STANDBY_INSTANCE = db2inst
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
HEARTBEAT_INTERVAL(seconds) = 25
HADR_TIMEOUT(seconds) = 100
TIME_SINCE_LAST_RECV(seconds) = 3
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

```

## MON\_GET\_HADR table function

If you issue this query on the primary, it will return information on all standbys. If you want to issue the MON\_GET\_HADR function against a standby database, be aware of the following points:

- You must enable reads on standby on the standby.
- If your HADR setup has multiple standby databases, the table function does not return any information about any other standbys.
- HADR in a Db2 pureScale environment does not support reads on standby, so the table function cannot be used on a standby.

For example, you could issue the following query on the primary database:

```

db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE,
          varchar(PRIMARY_MEMBER_HOST,20) as PRIMARY_MEMBER_HOST,
          varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST
from table (mon_get_hadr(NULL))"

```

Sample output is as follows:

HADR_ROLE	STANDBY_ID	HADR_STATE	PRIMARY_MEMBER_HOST	STANDBY_MEMBER_HOST
PRIMARY	1	PEER	hostP.ibm.com	hostS1.ibm.com

1 record(s) selected.

## GET SNAPSHOT FOR DATABASE command

This command collects status information and formats the output. The information that is returned is a snapshot of the database manager operational status at the time that you issued the command. HADR information is displayed in the output under the heading HADR status.

## db2GetSnapshot API

This API collects database manager monitor information and writes it to a user-allocated data buffer. The information that is returned is a snapshot of the database manager operational status at the time that the API was called.

### Other snapshot administrative views and table functions

You can use the following snapshot administrative views and table functions, which are not HADR specific and return a wider set of information, to query a subsection of the HADR information:

- ADMIN\_GET\_STORAGE\_PATHS
- MON\_GET\_TRANSACTION\_LOG
- MON\_GET\_DATABASE
- MON\_GET\_MEMORY\_POOL
- MON\_GET\_MEMORY\_SET
- MON\_GET\_TRANSACTION\_LOG

## Responding to an unplanned outage

If your database management software or cluster management software detects that a database server has failed, your database solution must respond to that failure as quickly and as smoothly as possible.

Your database solution must attempt to shield user applications from the failure by rerouting workload, if possible, and failover to a secondary or standby database, if one is available.

## Procedure

If your database or cluster management software detects that a database server has failed, you or your database or cluster management software must do the following:

1. Identify, bring online, and initialize a secondary database server to take over operations for the failed database server.

If you are using Db2 High Availability Disaster Recover (HADR) to manage primary and standby database servers, HADR will manage keeping the standby database synchronized with the primary database; and HADR will manage the takeover of the primary database by the standby database.

2. Reroute user application workload to the secondary database server.

Db2 client reroute can automatically reroute client application away from a failed database server to a secondary database server previously identified and configured for this purpose.

3. Remove the failed database server from the system to repair it.

Once the user applications have been rerouted to a secondary or standby database server, the failed database server can not handle any client application requests until it has been restarted or otherwise repaired. For example, if the cause of the failure on the primary database was that a database instance terminated unexpectedly, the Db2 fault monitor facility will automatically restart it.

## Performing an HADR failover operation:

When you want the current standby database to become the new primary database because the current primary database is not available, you can perform a forced takeover, or *failover*.

### Before you begin

A takeover operation can only take place if the standby is in peer state, disconnected peer state, remote catchup pending state, or remote catchup state. If the standby database is in any other state, an error will be returned.

**Note:** You can make a standby database that is in local catchup state available for normal use by converting it to a standard database. To do this, shut the database down by issuing the **DEACTIVATE DATABASE** command, and then issue the **STOP HADR** command. Once HADR has been stopped, you must complete a rollforward operation on the former standby database before it can be used. A database cannot rejoin an HADR pair after it has been converted from a standby database to a standard database. To restart HADR on the two servers, follow the procedure for initializing HADR.

If you have configured a peer window, shut down the primary before the window expires to avoid potential transaction loss in any related failover.

### About this task

The **TAKEOVER HADR** command with the **BY FORCE** can only be issued on the standby database. In Db2 pureScale environments, you can issue the command from any member in the standby cluster, including non-replay members.

During a failover, the following steps occur:

1. A disabling message is sent to the primary, if it is connected.
2. After it receives the disabling message, the primary database is shut down and log writing is stopped.
3. Log shipping and log retrieval is stopped on the standby, which entails a risk of data loss.
4. All received logs (that is, the logs that are stored in the log path) are replayed on the standby.
5. Any open transactions are rolled back on the standby.
6. The standby's role changes to primary and the new primary database is opened for client connections.

### Warning:

This procedure might cause a loss of data. Review the following information before performing this emergency procedure:

- Ensure that the primary database is no longer processing database transactions. If the primary database is still running, but cannot communicate with the standby database, executing a forced takeover operation (issuing the **TAKEOVER HADR** command with the **BY FORCE** option) could result in two primary databases. When there are two primary databases, each database will have different data, and the two databases can no longer be automatically synchronized.
  - Deactivate the primary database or stop its instance, if possible. (This might not be possible if the primary system has hung, crashed, or is otherwise

inaccessible.) After a failover operation is performed, if the failed database is later restarted, it will not automatically assume the role of primary database.

- The likelihood and extent of transaction loss depends on your specific configuration and circumstances:
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is synchronous (SYNC), the standby database does not lose transactions that were reported committed to an application before the primary database failed.
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is near synchronous (NEARSYNC), the standby database can only lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is asynchronous (ASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if the standby database crashes before it was able to write all the received logs to disk.

**Note:** Peer window is not allowed in ASYNC mode, therefore the primary database can never enter disconnected peer state in that mode.

- If the primary database fails while in remote catchup state and the synchronization mode is super asynchronous (SUPERASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if the standby database crashes before it was able to write all the received logs to disk.

**Note:** Databases can never be in peer or disconnected peer state in SUPERASYNC mode.

- If the primary database fails while in remote catchup pending state, transactions that have not been received and processed by the standby database are lost.

**Note:** Any log gap shown in the database snapshot represents the gap at the last time the primary and standby databases were communicating with each other; the primary database might have processed a very large number of transactions since that time.

- Ensure that any application that connects to the new primary (or that is rerouted to the new primary by client reroute), is prepared to handle the following:
  - There is data loss during failover. The new primary does not have all of the transactions committed on the old primary. This can happen even when the **hadr\_syncmode** configuration parameter is set to SYNC. Because an HADR standby applies logs sequentially, you can assume that if a transaction in an SQL session is committed on the new primary, all previous transactions in the same session have also been committed on the new primary. The commit sequence of transactions across multiple sessions can be determined only with detailed analysis of the log stream.
  - It is possible that a transaction can be issued to the original primary, committed on the original primary and replicated to the new primary (original standby), but not be reported as committed because the original primary crashed before it could report to the client that the transaction was

committed. Any application you write should be able to handle that transactions issued to the original primary, but not reported as committed on the original primary, are committed on the new primary (original standby).

- Some operations are not replicated, such as changes to database configuration and to external UDF objects.
- HADR does not interface with the Db2 fault monitor (db2fm) which can be used to automatically restart a failed database. If the fault monitor is enabled, you should be aware of possible fault monitor action on a presumably failed primary database.

## Procedure

To fail over the primary role to the standby:

- Use the CLP to initiate a failover operation on the standby database.
  1. Completely disable the failed primary database. When a database encounters internal errors, normal shutdown commands might not completely shut it down. You might need to use operating system commands to remove resources such as processes, shared memory, or network connections.
  2. Issue the **TAKEOVER HADR** command with the **BY FORCE** option on the standby database. In the following example the failover takes place on database LEAFS:

```
TAKEOVER HADR ON DB LEAFS BY FORCE
```

The **BY FORCE** option is required because the primary is expected to be offline.

If the primary database is not completely disabled, the standby database still has a connection to the primary and sends a disabling message to the primary database forcing it to shut down. The standby database still switches to the role of primary database whether or not it receives confirmation from that the primary database has been shutdown.

- Call the db2HADRTakeover application programming interface (API) from an application.
- Open the task assistant for the **TAKEOVER HADR** command in IBM Data Studio.

## Results

If, at the time of the failover, the standby has a connection to the primary (or any member on the primary in a Db2 pureScale environment), it sends a disabling message to the old primary to prevent a *split brain* scenario with dual primaries. You can clear the disabling message by doing one of the following:

- starting the failed primary as a standby (that is, reintegrating it)
- starting the failed primary as a primary using the **BY FORCE** option
- stopping HADR on the failed primary
- dropping the failed primary database
- restoring the database

## What to do next

If you want to reintegrate the old primary as the new standby, the old primary's log streams cannot have diverged from the new primary's. For more information on this procedure, see the Related links.

## Switching database roles in high availability disaster recovery (HADR):

You can initiate a takeover operation on a high availability disaster recovery (HADR) standby to switch the roles of the primary and standby databases.

### Before you begin

You can only perform a role switch between the primary and standby databases if the databases are in one of the following states (for Db2 pureScale environments, if every stream meets one of these conditions):

- peer state
- remote catchup state, when the synchronization mode is SUPERASYNC
- assisted remote catchup state (Db2 pureScale environments only)

If the standby database is in any other state, an error message is returned. If member or group crash recovery is in progress on the primary, the takeover operation fails.

### About this task

The **TAKEOVER HADR** command can only be issued on the standby database. If the primary database is not connected to the standby database when the command is issued, the takeover operation fails. In Db2 pureScale environments, you can issue the command from any member in the standby cluster, including non-replay members.

During a role switch, the following occurs on the primary:

1. New connections are rejected, open transactions are rolled back, and all remaining logs are shipped to the standby.
2. The primary changes to the standby role, and log receiving and replay is started.

And the following occurs on the standby:

1. After it has been confirmed that the old primary is now in the standby role, the standby changes its role to primary.
2. Log receiving is stopped after the end of logs. This ensures no data loss.
3. All received logs are replayed.
4. The new primary database is opened for client connections.

### Procedure

To switch the HADR database roles:

- Use the CLP to initiate a takeover operation on the standby database by issuing the **TAKEOVER HADR** command without the **BY FORCE** option on the standby database.
- Call the db2HADRTakeover application programming interface (API) from an application.
- Open the task assistant for the **TAKEOVER HADR** command in IBM Data Studio.

### Example

In the following example, the takeover operation takes place on the standby database LEAFS:

```
TAKEOVER HADR ON DB LEAFS
```

A log full error is slightly more likely to occur immediately following a takeover operation. To limit the possibility of such an error, an asynchronous buffer pool flush is automatically started at the end of each takeover. The likelihood of a log full error decreases as the asynchronous buffer pool flush progresses. Additionally, if your configuration provides a sufficient amount of active log space, a log full error is even more unlikely. If a log full error does occur, the current transaction is aborted and rolled back.

**Note:** Issuing the **TAKEOVER HADR** command without the **BY FORCE** option causes any applications currently connected to the HADR primary database to be forced off. This action is designed to work in coordination with automatic client reroute to assist in rerouting clients to the new HADR primary database after a role switch. However, if the forcing off of applications from the primary would be disruptive in your environment, you might want to implement your own procedure to shut down such applications prior to performing a role switch, and then restart them with the new HADR primary database as their target after the role switch is completed.

### Reintegrating a database after a takeover operation

If you issued a takeover operation in a high availability disaster recovery (HADR) environment because the primary database failed, you can bring the failed database back online and use it as a standby database or return it to its status as primary database.

#### Before you begin

Reintegration succeeds only if the old primary's log streams did not diverge from the new primary's; in a Db2 pureScale environment, if any log stream diverges, reintegration fails. In particular, HADR requires that the original primary database did not apply any logged operation that was never reflected on the original standby database before it took over as the new primary database. If this divergence did occur, you can restart the original primary database as a standby database by restoring a backup image of the new primary database or by initializing a split mirror.

Normally, failover is performed only when the old primary is offline. But in some scenarios, it might be still online (in a Db2 pureScale environment, it might be online on some members). For example, it might have become inaccessible to clients, making a failover necessary. Do not run the **DEACTIVATE DATABASE** command on the old primary before reintegration because deactivation modifies the log stream, making the old primary's log stream incompatible to the new primary's. Instead, kill any remaining members on the old primary.

Because it might be necessary to kill an old primary database, it is recommended that only one HADR database be created in a Db2 instance, so that the kill does not impact other databases.

In a Db2 pureScale environment, the database must be offline on all of the old primary's members. The CF does not need to be shut down and restarted; any leftover CF data structure for this database is cleaned upon reintegration.

#### About this task

Successful return of the **START HADR** command does not indicate that reintegration succeeded; it means only that the database started. Reintegration is still in progress. If reintegration subsequently fails, the database shuts itself down. You



should monitor standby states by using the **db2pd** command or the **MON\_GET\_HADR** table function to make sure that the standby database stays online and proceeds with the normal state transition. If necessary, you can check the administration notification log file and the **db2diag** log file to find out the status of the database.

## Procedure

To reintegrate the failed primary database into the HADR pair as the new standby database:

1. Repair the system where the original primary database is located. This could involve repairing failed hardware or rebooting the crashed operating system.
2. Restart the failed primary database as a standby database. In the following example, database LEAFS is started as a standby database:

```
START HADR ON DB LEAFS AS STANDBY
```

**Note:** In a Db2 pureScale environment, make sure that you issue the command from the member that you want to designate as the preferred replay member

## What to do next

After the original primary database rejoins the HADR pair as the standby database, you can choose to initiate a failback operation to switch the roles of the databases and return the original primary database to its initial role. To perform this failback operation, issue the following command on the standby database:

```
TAKEOVER HADR ON DB LEAFS
```

### Note:

1. If the HADR databases are not in peer state or the pair is not connected, this command fails.
2. Open sessions on the primary database are forced closed and inflight transactions are rolled back.
3. When you are switching the roles of the primary and standby databases, you cannot specify the **BY FORCE** option of the **TAKEOVER HADR** command.

---

## Failure management with Db2 cluster services

This section contains an overview of Db2 cluster services and detailed information about how it deals with specific types of host, cluster caching facility, and member failure.

Db2 cluster services is software that provides automatic heartbeat failure detection and automatically initiates the necessary recovery operations after a failure is detected. It also provides the cluster file system that gives each host in a Db2 pureScale instance access to a common file system. Db2 cluster services includes technology from IBM Tivoli System Automation for Multiplatforms (Tivoli SA MP) software, IBM Reliable Scalable Clustering Technology (RSCT) software, and IBM General Parallel File System (GPFS) software. This technology is packaged as an integral part of the Db2 pureScale Feature.

## Automated cluster caching facility failover

If the primary cluster caching facility (CF) fails, Db2 cluster services automatically attempts to restart it and fails over the primary role to the secondary cluster caching facility (assuming the recommended two-cluster caching facility setup).

Because of the integral role the cluster caching facility plays in a Db2 pureScale instance, the failed cluster caching facility is restarted and reintegrated as quickly as possible. The affect of the failure depends on these factors:

- How many cluster caching facilities are in the Db2 pureScale instance

If there is a single cluster caching facility in the Db2 pureScale instance, the instance will be brought down. If the cluster caching facility failed because of a software failure, a group restart will be initiated automatically. If the cluster caching facility failed because of a hardware failure, the user needs to fix the problem, after which a group restart will be initiated automatically.

If there are two cluster caching facilities in the Db2 pureScale instance, which is the recommended setup, Db2 cluster services attempts to fail over the primary role to the secondary cluster caching facility. If the primary cluster caching facility failed because of a software failure, it will be restarted automatically and reintegrated as the secondary cluster caching facility. If the primary cluster caching facility failed because of a hardware failure, the user needs to fix the problem, and then it will be restarted automatically and reintegrated as the secondary cluster caching facility.
- What the state of the secondary cluster caching facility is when the primary cluster caching facility fails

If the secondary cluster caching facility is in PEER state, Db2 cluster services will fail over the primary role to it.

If the secondary cluster caching facility is not in PEER state, the instance will be brought down. Db2 cluster services then initiates a group restart with the former secondary cluster caching facility now in the primary role.

## Automated restart

In a Db2 pureScale environment, Db2 cluster services automatically detects software and hardware failures, and initiates either a member restart or a group restart, depending on the type of failure that has occurred.

Automated restart helps ensure that host, member, or cluster caching facility failures have a minimal affect on the database. A member failure (and the subsequent restart) is transparent to applications and only temporarily affects uncommitted transactions running on the failed member. Applications can continue to access databases despite multiple host or member failures.

In situations where Db2 cluster services cannot restart the failed member on its original host, the failed member is restarted on another host in a process known as *restart light*. Some extreme failures, such as the loss of all cluster caching facilities in the Db2 pureScale instance, do result in a database outage. In these cases, Db2 cluster services initiates a group restart of the cluster caching facilities and members.

## Member restart and crash recovery

*Member restart* is the process of restarting the database server processes on a failed member and performing member crash recovery on each database that requires it.

If a software or hardware failure on a host causes a member to fail, Db2 cluster services detects the failure and automatically restarts the member. The member restart can either be a *local restart*, meaning that it is restarted on its original host (*home host*) or a *restart light*, meaning it is restarted on a different host:

### Local restart

If a member fails because of a software failure but the member's home host is still active, Db2 cluster services attempts a local restart. The local

member restart uses a reduced memory model to ensure the in-flight data is recovered to a consistent state quickly. Once the in-flight data has been recovered, the database's normal memory model is then initialized for full transaction processing.

### **Restart light**

If a member's home host is inactive or if a local restart attempt fails, the member is automatically restarted as a *guest member* on another member's home host using a minimal amount of resources. A member running in restart light mode does not process new transactions because its sole purpose is to perform member crash recovery.

Multiple member failures are generally recoverable with independent concurrent member restart recoveries, and, as a result, a group restart is not usually required. The database continues to remain open for access through other surviving members. Only data that was in-flight on the failed members is unavailable for the duration of the member restarts.

### **Member crash recovery**

Member crash recovery is responsible for the rolling back of incomplete transactions and the completing of committed transactions as a part of member restart. This will ensure that the database data modified by this member is brought to a consistent state. If indoubt transactions exist at the end of member crash recovery, the member will be made available to resolve them.

Member crash recovery will be performed when a viable cluster caching facility is still available and the database on a member is activated and it is determined that the member's log stream is inconsistent on disk due to an abnormal termination. In most cases, member crash recovery will be automatically invoked through member restart and the automatic recovery agent. The automatic recovery agent that is started when the instance is brought up, takes action when it detects that a database on the member is inconsistent.

Once member crash recovery of a database completes, the member will be able to accept incoming connection requests from other applications if that member was restarted on its original host.

### **Group restart and crash recovery**

*Group restart* is the process of restarting the entire Db2 pureScale instance by restarting the database server processes for all members and cluster caching facilities and performing a group crash recovery to bring the database back online.

A group restart occurs when there is not a viable primary cluster caching facility in the Db2 pureScale instance. This event is automatically detected and handled by Db2 cluster services. Group restart will be automatically initiated as soon as a primary cluster caching facility and a member become available. As group restart occurs, the database will be inaccessible across all members.

There are a few situations that can lead to the need for a group restart:

- If the instance is running with only one cluster caching facility, and that cluster caching facility fails.
- The primary cluster caching facility fails before the secondary cluster caching facility has reached PEER state.
- If both cluster caching facilities fail.

## Group crash recovery

Group crash recovery is responsible for making the database consistent by redoing any work that had not been written to disk and rolling back any transactions that had not been committed at the time of the failure. Group crash recovery is similar to Db2 crash recovery outside of a Db2 pureScale environment, but it uses the merged log streams from all members active on the database. Because group crash recovery automatically occurs as part of group restart, users generally do not have to take any action if a group crash recovery is required while a functioning cluster manager is present.

## Restart light

When a failed member cannot be restarted on its original host, or *home host*, Db2 cluster services restarts that member on one of the other available hosts in the Db2 pureScale instance. This process, known as *restart light*, allows member crash recovery to be performed without significantly affecting the rest of the instance.

A member that has been restarted in restart light mode on another host is known as a *guest member*, whereas a member that is running on its home host is known as a *resident member*. A guest member uses fewer resources than a resident member, and it cannot accept instance attachments or a database connections from an external application (SQL1032N).

The sole purpose of starting a member in restart light mode is to perform member crash recovery. The guest member is restarted using a pre-allocated, reduced memory model to minimize the affect on the resident member whose host is used for the restart light. After the guest member running in restart light mode completes member crash recovery on all required databases, it waits to be failed back to its home host. It cannot process new transactions until it is failed back to its home host.

**Note:** A member in restart light mode appears to be up and running (but in `WAITING_FOR_FAILBACK` state) if its state is queried. When the failed home host comes back online, the member in restart light mode will automatically be failed back to its own home host by Db2 cluster services and will become a fully active member that can process new transactions again.

Restart light is an automated process that works as follows: For software failures where the home host of the member is still active, Db2 cluster services attempts to restart a failed member on its home host. If the first restart attempt of the failed member does not succeed on its home host, that member is restarted in light mode on a different host as a guest member.

Restart light is immediately initiated for a member if one of the following situations occurs:

- a network failure where the home host loses its connection with the rest of the Db2 pureScale instance
- the home host becomes inactive unexpectedly because of a power failure or a hardware, LPAR, VM, or OS reset
- the member terminates abnormally when its home host is being stopped for maintenance

A restart light is a very rapid process because there is a set of Db2 idle processes on each host that preallocates resources for the restarting of guest members. Db2 will activate these processes instead of creating new processes to perform member

restart in light mode. As new processes are not created during restart light and because the guest member does not compete with the resident member for resources, member recovery processing is sped up.

### Disabling automatic member failback:

In some situations, you might want to delay the automatic failback, this is done through the **db2cluster** command.

### About this task

By default, Db2 cluster services fails back any member running in restart light mode as soon as that home host of member becomes available and any alerts are cleared. If you want to investigate the cause of a host failure, it can be useful to disable automatic member failback until a more suitable time.

### Procedure

To disable automatic failback:

1. Log in as an instance user and issue the **db2cluster** command:  
`db2cluster -cm -set -option autofailback -value off`
2. Restart the Db2 pureScale instance.

### Results

Any member that is in restart light mode and any future restart light members will remain on their guest host even if all member alerts have been cleared and the resident host of the member is active.

### Example

After experiencing a repeated host failure that resulted in members being restarted in light mode, the DBA decides that it is better to keep any failed hosts free of their resident members until the cause of the failure can be determined. The DBA issues the command:

```
db2cluster -cm -set -option autofailback -value off
```

This message is returned:

```
The db2cluster command succeeded. The AUTOFAILBACK option has been set to "OFF". Automatic failback will be disabled the next time that the Db2 database manager instance is restarted.
```

**Note:** If, for some reason, automatic failback had already been disabled, the command would have successfully completed, but the message will not mention restarting the instance.

The DBA restarts the instance. After investigating the cause of the host failure, the DBA issues the command to turn on the automatic member failback:

```
db2cluster -cm -set -option autofailback -value on
```

This message is returned:

```
The db2cluster command succeeded. The AUTOFAILBACK option has been set to "ON". Automatic failback will be enabled the next time that the Db2 database manager instance is restarted.
```

After restarting the instance, the DBA ensures that automatic member failback is enabled using the command:

```
db2cluster -cm -list -autofailback
```

This message is returned:  
The AUTOFAILBACK option is set to "ON".

**Memory considerations for restart light:**

A limited amount of memory is reserved for recovery purposes when Db2 is started, to facilitate the recovery of members in restart light mode. This reserved restart light memory is predefined, thus improving recovery performance, as the memory is reserved and immediately ready for use during recovery.

The amount of memory that can be reserved for restart light recovery purposes on a given host is limited by the *rstrt\_light\_mem* database manager configuration parameter. The default value of *rstrt\_light\_mem* is AUTOMATIC, which means that Db2 automatically calculates a fixed upper bound for the amount of memory to be pre-allocated and reserved for restart light recovery purposes and sets the value when Db2 is started. Db2 calculates the value based on the settings for the *instance\_memory* and *numdb* configuration parameters and the number of members on the host. The automatically calculated value ranges between 1 and 10 percent of the instance memory limit and is included in the total amount of instance memory. However, because the amount of reserved restart light memory can affect the performance of a resident member, users can adjust the restart light memory configuration to be appropriate for their specific workloads.

**Displaying the reserved restart light memory**

To display information about the total amount of memory allocated on a Db2 host, use the **db2pd** command with the **-totalmem** option. This information includes the amount of reserved restart light memory that is preallocated on the current Db2 host being accessed. To retrieve information for all hosts in a cluster, run db2pd on separate hosts in parallel. In the following example, db2pd is run on Host B, which has member 20.

```
db2pd -totalmem
```

	Controller	Memory	Current	HWM	Cached
	Automatic	Limit	Usage	Usage	Memory
Member 20	Yes	25750080 KB	9031201 KB	9391744 KB	480064 KB
<b>Restart Light Memory</b>	<b>Yes</b>	<b>2575008 KB</b>	<b>64182 KB</b>	<b>69265 KB</b>	<b>5250 KB</b>

Total current usage: 9095383 KB  
Total cached memory: 485314 KB

**Recovery of hidden buffer pools**

For member crash recovery, a reduced memory model is used for the buffer pools. As the buffer pools typically use up the largest amount of memory in the database shared memory set, the allocation of large buffer pools is very time consuming. The reduced memory model improves the recovery performance because small recovery hidden buffer pools are allocated instead of large user-defined buffer pools, which are very expensive. Just as with the existing hidden buffer pools, there are four recovery hidden buffer pools, one of each size 4K, 8K, 16K, and 32K. However, the hidden buffer pools are always 16 pages in size; the recovery hidden buffer pools have a minimum size of 250 pages and can be larger, depending on the restart light memory set size and the buffer pool size calculations.

In the following example, two user buffer pools, BP1 with 100 pages and BP2 with 200 pages, have been created for database TESTDB. Member 0 is in restart light

mode and Member 1 is not in restart light mode. The example includes a portion of the output from the following db2pd command. Member 1 shows the user-created buffer pools and the hidden buffer pools, although Member 0 only shows the 4 recovery hidden buffer pools.

```
db2pd -allmembers -db testdb -bufferpools
```

```
Database Member 1--Database TESTDB--Active--Up 0 days 00:00:14--Date 08/12/2010 18:55:19
```

Bufferpools:

```
First Active Pool ID      1
Max Bufferpool ID         3
Max Bufferpool ID on Disk 3
Num Bufferpools           7
```

Address	Id	Name	PageSz	PA-NumPgs	BA-NumPgs	BlkSize
0x00002AAAE443140	1	IBMDEFAULTBP	4096	1000	0	0
0x00002AAAE45B080	2	BP1	4096	100	0	0
0x00002AAAE45F060	3	BP2	4096	200	0	0
0x00002AAADB83CC0	4096	IBMSYSTEMBP4K	4096	16	0	0
0x00002AAADB13CC0	4097	IBMSYSTEMBP8K	8192	16	0	0
0x00002AAADB03CC0	4098	IBMSYSTEMBP16K	16384	16	0	0
0x00002AAAE453140	4099	IBMSYSTEMBP32K	32768	16	0	0

...NumTbsp	PgsToRemov	CurrentSz	PostAlter	SuspndTSCt	Automatic
3	0	1000	1000	0	False
0	0	100	100	0	False
0	0	200	200	0	False
0	0	16	16	0	False
0	0	16	16	0	False
0	0	16	16	0	False
0	0	16	16	0	False

```
Database Member 0 -- Database TESTDB -- Active -- Up 0 days 00:00:13
```

Bufferpools:

```
First Active Pool ID      4096
Max Bufferpool ID         0
Max Bufferpool ID on Disk 3
Num Bufferpools           4
```

Address	Id	Name	PageSz	PA-NumPgs	BA-NumPgs	BlkSize
0x00002AAAD9F946E0	4096	IBMSYSTEMBP4K	4096	9954	0	0
0x00002AAADA743140	4097	IBMSYSTEMBP8K	8192	250	0	0
0x00002AAADA733140	4098	IBMSYSTEMBP16K	16384	250	0	0
0x00002AAADA74B080	4099	IBMSYSTEMBP32K	32768	250	0	0

...NumTbsp	PgsToRemov	CurrentSz	PostAlter	SuspndTSCt	Automatic
3	0	9954	9954	0	False
0	0	250	250	0	False
0	0	250	250	0	False
0	0	250	250	0	False

## Memory consumption during a restart light

Ideally, a restart allows the prompt recovery of a failed member on a host other than the home host of the member without affecting that host's resident member. To achieve this, the reserved recovery memory for restart light is used first to perform database recovery operations. However, if the database recovery requires memory resources that exceed the restart light memory allocation, the restart light makes additional memory requests for free instance memory. These critical memory requests attempt to reduce current memory usage by the resident member. If there are still insufficient memory resources to finish the restart light process, Db2 requests additional memory from the operating system. If this occurs, all other noncritical memory requests fail until enough memory has been freed for the recovery operation. Applications running on the resident member can get out-of-memory failures, but the resident member still stays up. Once the recovery is completed, and the database or databases are consistent, any additional memory used by the guest member, beyond the originally reserved recovery memory, is freed.

These additional requests for memory resources for a restart light are temporary, but they can have a negative affect on the resident workload of the member. If you find that the reserved recovery memory is insufficient, consider increasing the size of the *rstrt\_light\_mem* database manager configuration parameter. The parameter is configurable but not online, so any changes require either a global **db2stop** and **db2start** or, if you want to update *rstrt\_light\_mem* on a member-by-member basis (that is, you do not want to stop all of the members at the same time) you must stop and start each member and the instance on each host of the member as follows:

```
db2 update dbm cfg using RSTRT_LIGHT_MEM 5
```

```
db2stop member 10
db2stop instance on hostA.torolab.ibm.com
db2start instance on hostA.torolab.ibm.com
db2start member 10
```

```
db2stop member 20
db2stop instance on hostB.torolab.ibm.com
db2start instance on hostB.torolab.ibm.com
db2start member 20
```

### Displaying the restart light memory consumption

There are two ways to display information about the total amount of memory being used on a host by the resident members and guest members:

1. Running the **db2pd** command with the **-totalmem** option on each host. Take the following example:

- Member 0 on Host A
- Member 1 on Host B
- Member 2 on Host C

Member 0 fails over to Host B in restart light mode. The user runs the **db2pd** command on Host B with resident member 1, as guest member 0 is running in restart light mode. Then the user runs the **db2pd** command on Host C to display the memory for member 2. Member 0 is labeled as "guest" in the display output and the memory usage is displayed in kilobytes. **db2pd** does not require a database connection.

Host B:

```
$ db2pd -totalmem
Total Memory Statistics in KB
```

	Controller Automatic	Memory Limit	Current Usage	HWM Usage	Cached Memory
Member 0 (guest)	Yes	20677572	242496	244032	15168
Member 1	Yes	20677572	186624	697088	46080
Restart Light Memory	Yes	839720	459392	459392	19200

```
Total current usage: 888512
Total cached memory: 80448
```

Host C:

```
$ db2pd -totalmem
Total Memory Statistics in KB
```

	Controller Automatic	Memory Limit	Current Usage	HWM Usage	Cached Memory
Member 2	Yes	20153284	4728832	4728832	1055104



Restart Light Memory	Yes	839720	689088	689088	28800
----------------------	-----	--------	--------	--------	-------

Total current usage: 5417920

Total cached memory: 1083904

2. Using the SQL interface to display the memory usage for all members including members in restart light mode. It only needs to be run from one host. However, this method requires a database connection so it can't be run from a member in restart light mode because restart light members do not accept connections. This display does not label a member as "guest" and the memory usage is displayed in bytes.

```
$ db2 'SELECT * FROM TABLE (SYSPROC.ADMIN_GET_MEM_USAGE()) AS T'
```

DBPARTITIONNUM	MAX_PARTITION_MEM	CURRENT_PARTITION_MEM	PEAK_PARTITION_MEM
1	21173833728	4605345792	4605345792
0	21173833728	248840192	249888768
2	20636962816	4651352064	4651352064

3 record(s) selected.

### Monitoring members in restart light:

You can monitor the recovery progress of a member in restart light mode by running the **LIST UTILITIES** command on any active member. The **LIST UTILITIES** command can return the global recovery status of all members including members in restart light mode.

```
LIST UTILITIES SHOW DETAIL
```

```
ID = 1
Type = MEMBER CRASH RECOVERY
Database Name = SAMPLE
Partition Number = 0
Description = Member Crash Recovery (Light Mode)
Start Time = 11/22/2007 15:20:05.646020
State = Executing
Invocation Type = Automated
Progress Monitoring:
  Estimated Percentage Complete = 0
  Phase Number [Current] = 1
    Description = Forward
    Total Work = 4193976 bytes
    Completed Work = 0 bytes
    Start Time = 11/22/2007 15:20:05.646121
  Phase Number = 2
    Description = Backward
    Total Work = 4193976 bytes
    Completed Work = 0 bytes
    Start Time = Not Started
```

There are a number of methods you can use to obtain a point-in-time view of your Db2 pureScale environment, including the DB2\_GET\_INSTANCE\_INFO table function, the DB2\_MEMBER administrative view, the **LIST INSTANCE** command, and the **db2instance** command. Using any one of these methods allows you to determine whether any members are in restart light mode, what host they are being recovered on, and what state of their restart light recovery they are currently in.

In this example, the the DB2\_MEMBER administrative view shows member 2 is being restarted in light mode on host so3.

```

SELECT ID,
       varchar(HOME_HOST,10) AS HOME_HOST,
       varchar(CURRENT_HOST,10) AS CURRENT_HOST,
       varchar(STATE,21) AS STATE,
       ALERT
FROM SYSIBMADM.DB2_MEMBER

```

ID	HOME_HOST	CURRENT_HOST	STATE	ALERT
1	so1	so1	STARTED	NO
2	so2	so3	RESTARTING	NO
3	so3	so3	STARTED	NO

3 record(s) selected.

### Scenario: Restart light:

This scenario describes the steps that occur during a member restart in light mode. It covers the most common case where there is a single host failure that causes that host's resident member to be automatically restarted as a guest member on another host that is still active. The scenario also covers how the guest member is failed back to its home host.

### Initial setup

There are six hosts (HostA, HostB, HostC, HostD, HostE, HostF) in the Db2 pureScale instance:

- Member 10 is running on HostA (its home host)
- Member 20 is running on HostB (its home host)
- Member 30 is running on HostC (its home host)
- Member 40 is running on HostD (its home host)
- cluster caching facility 128 (CF 128) is running on HostE
- cluster caching facility 129 (CF 129) is running on HostF

There is a set of Db2 idle processes for the instance on each host with pre-allocated memory that is reserved for restart light recovery purposes. Db2 cluster services monitors all the resources in the cluster.

The status information for the hosts, members, and CFs can be displayed by using the **LIST INSTANCE** command . At this point, the **LIST INSTANCE** command returns:

LIST INSTANCE

MEMBER_ID	TYPE	STATE	HOME_HOST	CURRENT_HOST	ALERT
10	MEMBER	STARTED	hostA	hostA	NO
20	MEMBER	STARTED	hostB	hostB	NO
30	MEMBER	STARTED	hostC	hostC	NO
40	MEMBER	STARTED	hostD	hostD	NO
128	CF	PRIMARY	hostE	-	NO
129	CF	PEER	hostF	-	NO

HOSTNAME	STATE	INSTANCE_STOPPED	ALERT
hostA	ACTIVE	NO	NO
hostB	ACTIVE	NO	NO
hostC	ACTIVE	NO	NO
hostD	ACTIVE	NO	NO
hostE	ACTIVE	NO	NO
hostF	ACTIVE	NO	NO

## Host failure

A power failure occurs on the HostA server. Db2 cluster services cannot restart member 10 on HostA so it restarts the member in light mode on the next available host: HostB.

At this point, the **LIST INSTANCE** command shows that member 10's state is now **RESTARTING** and its current host is now HostB, and the state of HostA is **INACTIVE** (note that the **INSTANCE\_STOPPED** field is not set because the instance was not manually stopped on HostA) and it has an alert:

LIST INSTANCE

MEMBER_ID	TYPE	STATE	HOME_HOST	CURRENT_HOST	ALERT
10	MEMBER	<b>RESTARTING</b>	hostA	<b>hostB</b>	NO
20	MEMBER	STARTED	hostB	hostB	NO
30	MEMBER	STARTED	hostC	hostC	NO
40	MEMBER	STARTED	hostD	hostD	NO
128	CF	PRIMARY	hostE	-	NO
129	CF	PEER	hostF	-	NO

HOSTNAME	STATE	INSTANCE_STOPPED	ALERT
hostA	<b>INACTIVE</b>	NO	<b>YES</b>
hostB	ACTIVE	NO	NO
hostC	ACTIVE	NO	NO
hostD	ACTIVE	NO	NO
hostE	ACTIVE	NO	NO
hostF	ACTIVE	NO	NO

## Waiting for failback

After the process model is started, member crash recovery is performed on each database that requires it. To check the progress of the member crash recovery, use the **LIST UTILITIES** command with the **SHOW DETAIL** option, as described in Monitoring members in restart light. After member crash recovery completes, member 10 waits to be failed back to HostA and will not be able to process any new transactions until then. There can be indoubt transactions that must be resolved as member 10 is waiting to be failed back.

At this point, the **LIST INSTANCE** command shows that member 10's state is now **WAITING\_FOR\_FAILBACK**:

LIST INSTANCE

MEMBER_ID	TYPE	STATE	HOME_HOST	CURRENT_HOST	ALERT
10	MEMBER	<b>WAITING_FOR_FAILBACK</b>	hostA	hostB	NO
20	MEMBER	STARTED	hostB	hostB	NO
30	MEMBER	STARTED	hostC	hostC	NO
40	MEMBER	STARTED	hostD	hostD	NO
128	CF	PRIMARY	hostE	-	NO
129	CF	PEER	hostF	-	NO

HOSTNAME	STATE	INSTANCE_STOPPED	ALERT
hostA	INACTIVE	NO	YES
hostB	ACTIVE	NO	NO

hostC	ACTIVE	NO	NO
hostD	ACTIVE	NO	NO
hostE	ACTIVE	NO	NO
hostF	ACTIVE	NO	NO

### Issue with host resolved

Power® is restored to HostA, so HostA becomes active in the Db2 pureScale instance again.

At this point, the **LIST INSTANCE** command shows that HostA is now active and the alert has been cleared:

LIST INSTANCE

MEMBER_ID	TYPE	STATE	HOME_HOST	CURRENT_HOST	ALERT
-----					
10	MEMBER	WAITING_FOR_FAILBACK	hostA	hostB	NO
20	MEMBER	STARTED	hostB	hostB	NO
30	MEMBER	STARTED	hostC	hostC	NO
40	MEMBER	STARTED	hostD	hostD	NO
128	CF	PRIMARY	hostE	-	NO
129	CF	PEER	hostF	-	NO

HOSTNAME	STATE	INSTANCE_STOPPED	ALERT
-----			
hostA	<b>ACTIVE</b>	NO	<b>NO</b>
hostB	ACTIVE	NO	NO
hostC	ACTIVE	NO	NO
hostD	ACTIVE	NO	NO
hostE	ACTIVE	NO	NO
hostF	ACTIVE	NO	NO

### Failing back to the home host

Db2 cluster services detects that HostA is active and automatically fails back member 10 to that host.

At this point, the **LIST INSTANCE** command shows that the state of member 10 is now **RESTARTING** and its current host is again HostA:

LIST INSTANCE SHOW DETAIL

MEMBER_ID	TYPE	STATE	HOME_HOST	CURRENT_HOST	ALERT
-----					
10	MEMBER	<b>RESTARTING</b>	hostA	<b>hostA</b>	NO
20	MEMBER	STARTED	hostB	hostB	NO
30	MEMBER	STARTED	hostC	hostC	NO
40	MEMBER	STARTED	hostD	hostD	NO
128	CF	PRIMARY	hostE	-	NO
129	CF	PEER	hostF	-	NO

HOSTNAME	STATE	INSTANCE_STOPPED	ALERT
-----			
hostA	ACTIVE	NO	NO
hostB	ACTIVE	NO	NO
hostC	ACTIVE	NO	NO
hostD	ACTIVE	NO	NO
hostE	ACTIVE	NO	NO

## Restarting on the home host

When member 10 successfully completes member restart on HostA, its state is changed to STARTED, and it can now process new transactions and accept user connections. At this point, the **LIST INSTANCE** command returns the same information about the Db2 pureScale instance as in Initial setup.

## Manual intervention in failure situations

In most failure situations, there is no need for users to take action because Db2 cluster services automatically initiates the required type of restart or crash recovery. However, there are certain situations in which you must manually initiate restart or crash recovery, or take steps for group crash recovery to be completed.

### Initiating group crash recovery

Group crash recovery is almost always performed automatically, with no user action required. You will only initiate group crash recovery manually in situations where the cluster manager has repeatedly failed in its attempt to perform group crash recovery, or it is not present (perhaps because cluster management was disabled), or when the **autorestart** configuration parameter is set to OFF.

### About this task

Only one member can perform group crash recovery on a database at a given time. If two members initiate a group crash recovery of the same database at the same time, the second member's command will wait until group crash recovery completes on the first member.

In the case where there are indoubt transactions on the group crash recovery member at the conclusion of the crash recovery operation, the member that performed the group crash recovery is left active and is able to accept new work; however, any data associated with the indoubt transactions is not accessible because the rows accessed by the indoubt transaction must be protected (that is, they are locked) until its eventual resolution. The fact that indoubt transactions exist on the group crash recovery member means that if the member fails before the indoubt transactions are resolved, a subsequent member crash recovery must occur when the database is next activated on the member.

In the case where a group crash recovery is issued on member A and it is found, in the course of the group crash recovery, that member B has associated with it, one or more indoubt transactions, member B remains inconsistent after the group crash recovery. This implies that before member B can be started after the group crash recovery completes, a member crash recovery must occur on member B. After the member crash recovery completes, member B is available to accept new connections. It should be noted that this subsequent member crash recovery completes very quickly because there is no redo or undo work required (the crash recovery is only required to load the indoubt transactions into the member's transaction table), and all of the locks required to protect the data affected by the indoubt transactions are already reserved for the member in the cluster caching facility. Additionally, this subsequent RESTART operation is, in most cases, hidden from the user because when AUTORESTART is enabled, the member crash recovery is performed automatically when the first connection is made to the database on the given member.

## Procedure

To manually initiate group crash recovery, issue the **RESTART DATABASE** command from one of the members.

## Results

When the group crash recovery completes, the member on which the command was run will be able to accept new connections and all data will be available with the exception of data that remains locked by the unresolved indoubt transactions. After the **RESTART DATABASE** command completes, a connection will be maintained to the database if the user has **CONNECT** privilege for the database. The database will not be activated on any member other than the member on which group crash recovery was run.

## Initiating member crash recovery

Member crash recovery is almost always performed automatically, with no user action required. You will only initiate member crash recovery manually in situations where the cluster manager has repeatedly failed in its attempt to perform crash recovery, or it is not present (perhaps because cluster management was disabled), or when the **autorestart** configuration parameter is set to **OFF**.

## About this task

If the Db2 pureScale instance has more than one member requiring member crash recovery, the cluster manager issues multiple member crash recoveries in parallel. This helps avoid hang situations caused by one failed member having a dependency on resources held by another failed member.

## Procedure

To manually initiate member crash recovery, issue the **RESTART DATABASE** command. When the **RESTART DATABASE** command is issued, Db2 determines whether a group crash recovery or a member crash recovery is required.

## Results

Once member crash recovery completes, the database on the given member will be activated and available to receive connections from other applications.

## Recovering with damaged table spaces

If table space containers have been damaged and group crash recovery is required on the database, the group crash recovery operation will fail. In a Db2 pureScale environment with automatic recovery, you can resolve such a scenario as follows.

## Before you begin

This scenario demonstrates how to resolve the problem of damaged table spaces in a Db2 pureScale environment so that automatic group crash recovery can proceed.

## About this task

Assume that automatic recovery is active, but fails to recover the database because a table space has been damaged. After repeated failures, the Db2 member is moved to another host and restart light is attempted. After further failures to

automatically recover the database, the cluster manager raises an alert. The member that failed is not started and is inactive.

### Procedure

1. Disable automatic recovery by setting the **autorestart** configuration parameter to OFF. The **autorestart** configuration parameter is not dynamic and its new value takes effect the next time the member is started up. This command changes the **autorestart** configuration parameter to OFF for a database called WSDB:

```
UPDATE DATABASE CONFIGURATION FOR WSDB USING AUTORESTART OFF
```

2. Use the **db2cluster** command to clear all outstanding alerts for all members. Also, Db2 cluster services will automatically issue the **db2start** command to restart the members on their home hosts. The members that restart pick up the changed **autorestart** value and do not initiate automatic recovery.

```
db2cluster -clear -alert
```

3. Issue the **RESTART DATABASE** command to resolve the damaged table spaces. The damaged table spaces are marked as offline and put into the drop pending state.

```
RESTART DATABASE WSDB DROP PENDING TABLESPACES (tbsp1, tbsp2)
```

4. Re-enable automatic recovery by setting the **autorestart** configuration parameter to ON. For example, this command changes the **autorestart** configuration parameter to ON for the database called WSDB:

```
UPDATE DATABASE CONFIGURATION FOR WSDB USING AUTORESTART ON
```

The **autorestart** parameter is not dynamic and its new value takes effect the next time the member is started up.





---

## Chapter 2. Data recovery

Recovery is the rebuilding of a database or table space after a problem such as media or storage failure, power interruption, or application failure. If you have backed up your database, or individual table spaces, you can rebuild them should they become damaged or corrupted in some way.

There are four types of recovery:

- Crash recovery protects a database from being left in an inconsistent, or unusable, state when transactions (also called units of work) are interrupted unexpectedly.
- Disaster recovery consist of the process to restore a database in the event of a fire, earthquake, vandalism, or other catastrophic events.
- Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation.
- Rollforward recovery can be used to reapply changes that were made by transactions that were committed after a backup was made.

The Db2 database manager starts crash recovery automatically to attempt to recover a database after a power interruption. You can use version recovery or rollforward recovery to recover a damaged database.

---

### Developing a backup and recovery strategy

A database can become unusable because of hardware or software failure, or both. You might, at one time or another, encounter storage problems, power interruptions, or application failures, and each failure scenario requires a different recovery action.

Protect your data against the possibility of loss by having a well rehearsed recovery strategy in place.

Some of the questions that you should answer when developing your recovery strategy are:

- Will the database be recoverable?
- How much time can be spent recovering the database?
- How much time will pass between backup operations?
- How much storage space can be allocated for backup copies and archived logs?
- Will table space level backups be sufficient, or will full database backups be necessary?
- Should I configure a standby system, either manually or through high availability disaster recovery (HADR)?

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

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

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

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

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

Recovery log files and the recovery history file are created automatically when a database is created (Figure 10 on page 271). These log files are important if you need to recover data that is lost or damaged.

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

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

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

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

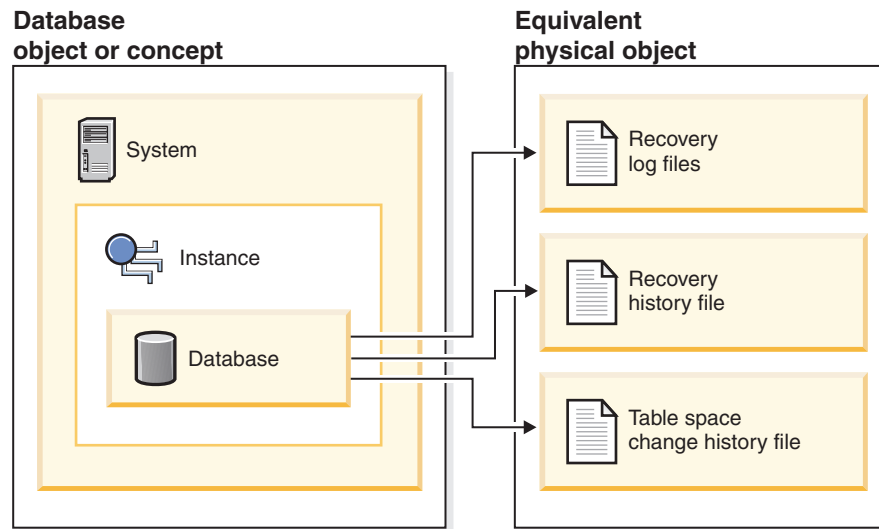


Figure 10. Database recovery files

Data that is easily re-created can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. If both the **logarchmeth1** and **logarchmeth2** database configuration parameters are set to OFF then the database is *Non-recoverable*. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken and rollforward recovery is not supported.

Data that *cannot* be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. *Recoverable databases* have the **logarchmeth1** or **logarchmeth2** database configuration parameters set to a value other than OFF. Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with rollforward recovery, you can roll the database forward (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

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

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table

space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

## Automated backup operations

Since it can be time-consuming to determine whether and when to run maintenance activities such as backup operations, you can use automatic maintenance. With automatic maintenance, you specify your maintenance objectives, including when automatic maintenance can run. Db2 then uses these objectives to determine if the maintenance activities need to be done and then runs only the required maintenance activities during the next available maintenance window (a user-defined time period for the running of automatic maintenance activities).

**Note:** You can still perform manual backup operations when automatic maintenance is configured. Db2 will only perform automatic backup operations if they are required.

## Deciding how often to back up

Your recovery plan should allow for regularly scheduled backup operations, because backing up a database requires time and system resources. Your plan might include a combination of full database backups and incremental backup operations. Also, the frequency and types of backups you make affect your database recovery time.

Take full database backups regularly, even if you archive the logs to allow for rollforward recovery. To recover a database, you can use either a full database backup image that contains all of the table space backup images, or you can rebuild the database by using selected table space images. Table space backup images are also useful for recovering from an isolated disk failure or an application error. In partitioned database environments, you need to restore only the table spaces that reside on database partitions that failed. You do not need to restore all of the table spaces or all of the database partitions.

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

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

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

## Online and offline backup considerations

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

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

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

## Selective table space backup considerations

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

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

If you keep your long field data, LOB data, and indexes in separate table spaces, but do not back them up together, consider the following point: 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.

## Table reorganization considerations

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.

## Table space modification status considerations

You can also make more informed decisions about whether to back up a table space by checking its modification status. The **db2pd -tablespaces trackmodstate** command and the **tbsp\_trackmode\_state** monitor element displays the status of the table space with respect to the last or next backup. You can use this information to determine whether the table space was modified or if the table space needs to be backed up.

## Database recovery time considerations

The time required to recover a database is made up of two parts:

- The time required to complete the restoration of the backup.

- If the database is enabled for forward recovery, the time required to apply the logs during the rollforward operation

When formulating a recovery plan, take these recovery costs and their impact on your business operations into account. Testing your overall recovery plan assists you in determining whether the time required to recover the database is reasonable, given your business requirements. Following each test, you might want to increase the frequency with which you take a backup. If rollforward recovery is part of your strategy, this increased backup frequency reduces the number of logs that are archived between backups and, as a result, reduces the time required to roll the database forward after a restore operation.

## Storage considerations for recovery

When deciding which recovery method to use, consider the storage space required. Backup and archived log file compression can help reduce the storage cost in your database environment.

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

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

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

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

- Specify an archived log file method using the LOGARCHMETH1 or LOGARCHMETH2 configuration parameters.
- Enable archived log file compression with the LOGARCHCOMPR1 and LOGARCHCOMPR2 configuration parameters.
- Manually copy the logs to a storage device or directory other than the database log path directory after they are no longer in the active set of logs.
- Use a user exit program to copy these logs to another storage device in your environment.

### Backup compression

In addition to the storage savings you can achieve through row compression in your active database, you can also use backup compression to reduce the size of your database backups.

Whereas row compression works on a table-by-table basis, when you use compression for your backups, *all* of the data in the backup image is compressed, including catalog tables, index objects, LOB objects, auxiliary database files and database meta-data.

You can use backup compression with tables that use row compression. Keep in mind, however, that backup compression requires additional CPU resources and extra time. It may be sufficient to use table compression alone to achieve a reduction in your backup storage requirements. If you are using row compression, consider using backup compression only if storage optimization is of higher priority than the extra time it takes to perform the backup.

**Tip:** Consider using backup compression only on table spaces that do not contain compressed data if the following conditions apply:

- Data and index objects are separate from LOB and long field data, and
- You use row and index compression on the majority of your data tables and indexes, respectively

To use compression for your backups, use the **COMPRESS** option on the **BACKUP DATABASE** command.

### Archived log file compression

As of Db2 V10.1, you can compress archived log files. This capability, in addition to data and index compression, along with backup compression, reduces the amount of disk space required for your database environment.

Archived log files are the third major space consumer for roll-forward recoverable databases. Archived log files contain a significant amount of data and these archives can grow quickly. If modified data is already in compressed tables, logging is reduced by virtue of including compressed record images in log records. Compression of archived log files further increases storage savings, even in these environments.

To use compression for your archived log files, you can use the **UPDATE DB CFG** command to set the **logarchcompr1** and **logarchcompr2** configuration parameters to ON.

### Restrictions

- Archived log file compression does not take effect under the following conditions.
  - The corresponding archived log file method is not set to DISK, TSM, or VENDOR. When the corresponding archived log file method is set as described, the log files are physically moved out of the active log path, or the mirror log path.
  - Whenever archived log file compression is enabled, but the corresponding log archiving method is set to OFF, LOGRETAIN or USEREXIT, archived log file compression has no effect. Any update to the **logarchmeth1** and **logarchmeth2** or the **logarchcompr1** and **logarchcompr2** database configuration parameters which results in such a scenario returns a warning, SQL1663W.

**Note:** When the database is activated, SQL1663W is not returned when setting or changing archived log file compression database configuration parameters. Instead, SQL1363W is returned, which is a higher priority message. If the database is not activated, the SQL1663W warning message is returned.

- Manual archiving and retrieval with **db2adut1**.

- The **db2adut1** utility does not perform compression or decompression during UPLOAD or EXTRACT operations. Movement of compressed log files to and from the archive location is fully supported by **db2adut1**.
- If logs are uploaded to Tivoli Storage Manager with **db2adut1**, and you want to compress archived log files, archived log file compression must be enabled when the logs are archived to the disk location, before **db2adut1** picks them up. If compressed logs are retrieved manually with **db2adut1**, they are decompressed on first access.
- Archived log file compression is not supported when raw devices are used for database logging.
  - Archived log file compression is not supported when either the **logpath** or the **newlogpath** database configuration parameters point to a raw device. Any database configuration update that results in archived log file compression being enabled while **logpath** or **newlogpath** database configuration parameters point to raw devices fails, SQL1665N.
- When enabling archived log file compression using the **logarchcompr1** and **logarchcompr2** database configuration parameters, logs already stored in a backup image are not affected.

## Hardware accelerated backup and log file compression

By using the nest accelerator NX842 of POWER 7+ and POWER 8 processors, you can achieve hardware compression for backup images and log archive files on AIX .

### Prerequisites

- This solution is only supported on AIX. Minimum AIX levels are AIX V7 TL3 SP3 and AIX V6 TL9 SP3.
- Active Memory Expansion (AME) has to be licensed but must not be enabled. This is a temporary restriction and not a technical limitation. In addition, Active Memory Sharing (AMS) has to be deactivated on the logical partition (LPAR).
- The CPU has to be a POWER 7+ or later.
- The following minimum firmware levels are recommended for POWER 8: FW820.50, FW830.30 or FW840.40.

**Remember:** Provided that the kernel requirements are met, it is possible to recover using the backup images and log files that were compressed with NX842 on previous POWER® versions.

### Advantages of using this solution

- A very fast compression can be achieved through the special hardware compression unit NX842 on POWER CPUs. The general CPU resources are not used for this compression.
- The NX842 compression units are typically not used for AME on database servers since deep row compression, adaptive compression and index compression can make memory compression inefficient.
- The compression algorithm in hardware provides faster compression than the common Db2 compression.

### How to use this as backup

To start a backup using the hardware compression, it is necessary to specify the library: backup database databasename compress comprlib libdb2nx842.a



The backups can be compressed by default with NX842. To achieve this the registry variable **DB2\_BCKP\_COMPRESSION** has to be set to NX842. Afterwards, issue the command: **backup database databasename compress**. The image will then be compressed using the NX842 hardware compression.

### Using the solution for log archive compression

The NX842 hardware compression can also be used for log archive compression. To activate this, change the database configuration parameter **LOGARCHCOMPR1** or **LOGARCHCOMPR2** to **NX842** using this command: **update database configuration for databasename using LOGARCHCOMPR1 NX842**

**Note:** These two parameters can still take different values. For example, the common Db2 compression can be used for **LOGARCHCOMPR1** and NX842 compression for **LOGARCHCOMPR2**:

```
update database configuration for databasename using LOGARCHCOMPR1 ON
update database configuration for databasename using LOGARCHCOMPR2 NX842
```

## Keeping related data together

You should group related data together to aid in data recovery.

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

- At the application level, when transactions update more than one table
- At the database level, where referential integrity exists between tables, or where triggers on one table affect another table.

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

## Backup and restore operations between different operating systems and hardware platforms

Db2 database systems support some backup and restore operations between different operating systems and hardware platforms.

The supported platforms for Db2 backup and restore operations can be grouped into one of three families:

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

A database backup from one platform family can only be restored on any system within the same platform family. For Windows operating systems, you can restore a database from a previous Db2 version on a later version. For Linux and UNIX operating systems, as long as the endianness (big endian or little endian) of the backup and restore platforms is the same, you can restore backups that were produced on down level versions.

The following table shows each of the Linux and UNIX platforms Db2 supports and indicates whether the platforms are big endian or little endian:

*Table 17. Endianness of supported Linux and UNIX operating systems Db2 supports*

Platforms	Support Restrictions	Endianness
AIX		Big endian
HP on IA64	Only supported for Db2 Version 10.5 images or lower	Big endian
Solaris SPARC	Only supported for Db2 Version 10.5 images or lower	Big endian
Linux on zSeries		Big endian
Linux on IBM Power Systems™	Only supported for Db2 Version 10.5 images or lower	Big endian
Linux on IBM Power Systems for Little Endian		Little endian
Linux on IA-64		Little endian
Linux on AMD64 and Intel EM64T		Little endian
32-bit Linux on x86	Only supported for Db2 Version 10.5 images or lower	Little endian

The target system must have the same (or later) version of the Db2 database product as the source system. You cannot restore a backup that was created on one version of the database product to a system that is running an earlier version of the database product. For example, you can restore a Db2 Version 10.1 on a Db2 Version 10.5 database system, but you cannot restore a Db2 Version 10.5 backup on a Db2 Version 10.1 database system.

**Note:** You can restore a database from a backup image that was taken on a 32-bit level into a 64-bit level, but not reversibly. The Db2 backup and restore utilities should be used to back up and restore your databases. Moving a file set from one machine to another is not recommended as this can compromise the integrity of the database.

In situations where certain backup and restore combinations are not allowed, you can move tables between Db2 databases using other methods:

- The **db2move** command
- The **export** command followed by the **import** or the **load** command

**Note:** Database configuration parameters are set to their defaults if the values in the backup are outside of the allowable range for the environment in which the database is being restored.

## Log stream merging and log file management in a Db2 pureScale environment

In a Db2 pureScale environment, each member maintains its own set of transaction log files (that is, a *log stream*) on the shared disk, each set in a separate log path. The log files for a member contain a history of all data changes that occurred on that member.

Multiple applications, each accessing a different member simultaneously, might generate dependent transactions during run time. A dependency between two transactions can occur if, for example, both transactions change the same row. To effectively interpret the log records, the Db2 data server must examine the records from all log streams and order the records so that they reflect the order of the updates that occurred at run time. This ordering is known as a *log stream merge* operation. Several operation types in a Db2 pureScale environment require log stream merges; these include (among others) group crash recovery, database roll-forward operations, and table space roll-forward operations.

## Logging configuration parameters in a Db2 pureScale environment

Table 18 shows which logging-related database configuration parameters are global in scope and which parameters are dynamically updatable.

*Table 18. Logging-related database configuration parameters*

Parameter	Global?	Dynamically updatable?
<b>archretrydelay</b>	Yes	Yes
<b>blk_log_dsk_ful</b>	No	Yes
<b>failarchpath</b>	Yes	Yes
<b>logarchcompr1</b>	Yes	Yes
<b>logarchcompr2</b>	Yes	Yes
<b>logarchmeth1</b>	Yes	Yes
<b>logarchmeth2</b>	Yes	Yes
<b>logarchopt1</b>	Yes	Yes
<b>logarchopt2</b>	Yes	Yes
<b>logbufsz</b>	No	Yes
<b>logfilsiz</b>	Yes	No
<b>logprimary</b>	Yes	No
<b>logsecond</b>	Yes	Yes
<b>max_log</b>	No	Yes
<b>mirrorlogpath</b> <sup>1</sup>	Yes	No
<b>newlogpath</b> <sup>1</sup>	Yes	No
<b>num_log_span</b>	No	Yes
<b>numarchretry</b>	Yes	Yes
<b>overflowlogpath</b>	Yes	Yes
<b>page_age_trgt_grc</b>	Yes	No
<b>page_age_trgt_mrc</b>	Yes	No
<b>softmax</b> <sup>2</sup>	Yes	No
<b>vendoropt</b>	Yes	Yes

Table 18. Logging-related database configuration parameters (continued)

Parameter	Global?	Dynamically updatable?
<sup>1</sup> The first member that connects to or activates the database processes the changes to this log path parameter. The Db2 database manager verifies that the path exists and that it has both read and write access to that path. It also creates member-specific subdirectories for the log files. If any one of these operations fails, the Db2 database manager rejects the specified path and brings the database online using the old path. If the database manager accepts the specified path, the new value is propagated to each member. If a member fails while trying to switch to the new path, subsequent attempts to activate the database or to connect to it fails, and SQL5099N is returned. All members must use the same log path.		
<sup>2</sup> <p><b>Important:</b> The <b>softmax</b> database configuration parameter is deprecated is deprecated in Version 10.5 and might be removed in a future release. For more information, see Some database configuration parameters are deprecated in <i>What's New for Db2 Version 10.5</i>.</p>		

## Retrieving logs for a log stream merge operation in a Db2 pureScale environment

A subdirectory is created in the path for retrieved log files. The subdirectory has the following format: *log\_path*/LOGSTREAMxxxx, where *log\_path* represents the log path, overflow log path, or mirror log path, and xxxx is a 4-digit log stream identifier. (The log stream identifier is not necessarily equivalent to the associated member ID.) Within this subdirectory, if a member requires log retrieval, the Db2 database manager creates another level of subdirectories for retrieved logs from each member. For example, if you specify an overflow log path of /home/dbuser/overflow/ on a 3-member system, and an application on member 0 must retrieve logs that are owned by other members, the path for member 0 is /home/dbuser/overflow/NODE0000/LOGSTREAM0000, and subdirectories under this path contain retrieved logs that are owned by other members, as shown in the following example:

```
Member 0 retrieves its own logs here:
/home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0000
Member 0 retrieves logs that belong to member 1 here:
/home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0001
Member 0 retrieves logs that belong to member 2 here:
/home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0002
```

**Note:** Do not manually insert log files in to these retrieve subdirectories. If you want to manually retrieve log files, use the overflow log path instead.

When reading archived log files that are owned by other members, a member might need to retrieve log files in to its own log path or overflow log path. In this case, the log stream merge operation creates a **db2logmgr** engine dispatchable unit (EDU) for each log stream, as needed.

As mentioned earlier, there are three paths that can be used to store log files that are owned by other members, as shown in the following list:

1. If you set the **overflowlogpath** database configuration parameter, the overflow log path is used.

**Tip:** You can use **ROLLFORWARD DATABASE** and **RECOVER DATABASE** command options to specify an alternative overflow log path; the values of these options override the database configuration for purposes of the single recovery operation.

2. The primary log path

3. If you set the **mirrorlogpath** database configuration parameter, the mirror log path is used.

If the Db2 database manager is unable to store a log file in the first path, it attempts to use the next path in the list. If none of these paths is available, the utility that invoked the log stream merge operation returns an error that is specific to that utility.

Output from the **GET DATABASE CONFIGURATION** command in a Db2 pureScale environment identifies each log path followed by the name of the member. For example, if the mirror log path was set to `/home/dbuser/mirrorpath/`, for member 2, the output displays `/home/dbuser/mirrorpath/NODE0000/LOGSTREAM0002`.

If you must manually retrieve log files that are owned by other members, ensure that the database manager can access the log files by using the same directory structure that is automatically created. For example, to make logs from member 2 available in the overflow log path of member 1, place the logs in the `/home/dbuser/overflow/NODE0000/LOGSTREAM0001/LOGSTREAM0002` directory.

Retrieved log files are automatically deleted when they are no longer needed. Subdirectories that were created during a log stream merge operation are retained for future use.

## Detection of missing logs during a log stream merge operation

If you accidentally deleted, moved, or archived and lost a log file that is required for a recovery operation, you can roll-forward recover the database to the last consistent point before the missing log file.

If, during a log stream merge operation, the Db2 database manager determines that there is a missing log file in one of the log streams, an error is returned. The roll-forward utility returns SQL1273N; the db2ReadLog API returns SQL2657N.

Figure 11 shows an example of how two members could write log records to the log files in their active log stream. Each log file is represented by a box.

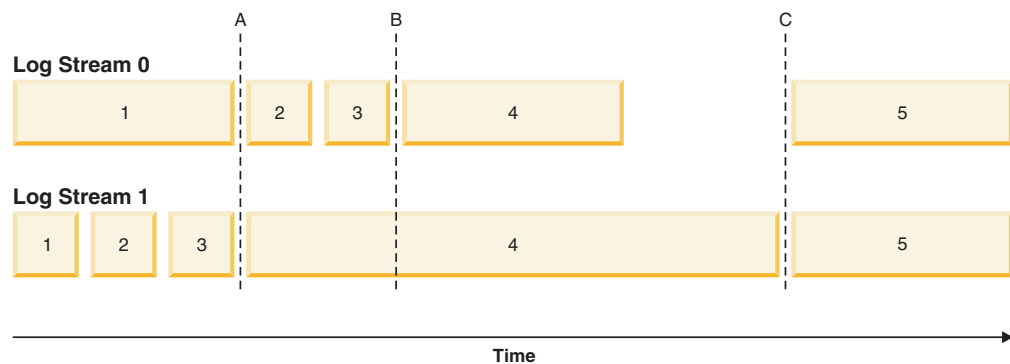


Figure 11. Log files in a Db2 pureScale environment

Consider a scenario where only log file 4 from log stream 1 is missing, a roll-forward operation to time A succeeds while roll-forward operations to time B, time C, or to the END OF LOGS fail. The ROLLFORWARD command returns SQL1273N because log file 4 is not available. Furthermore, since the log records in files 2 and 3 on log stream 0 were written during the same time period as the beginning of log file 4 on log stream 1, the roll-forward operation cannot process

log files 2 and 3 until log file 4 from log stream 1 is available. The result is that the roll-forward operation stops at time A, and any subsequent roll-forward operations cannot proceed beyond time A until log 4 from stream 1 becomes available.

Consider another scenario where only log file 4 from log stream 0 is missing during a roll-forward operation. If you issue a **ROLLFORWARD** command with the **END OF LOGS** option (or anytime after time B), the operation will stop at time B and will return SQL1273N because log file 4 on stream 0 is missing. A roll-forward operation can replay log records from files 2 and 3 on log stream 0 and some logs from file 4 on stream 1 up to time B. The roll-forward operation must stop at time B even though additional logs from stream 1 are available because the log merge process requires that all the logs from all the streams be available.

If you can find the missing log file, make it available and reissue the **ROLLFORWARD DATABASE** command. If you cannot find the missing log file, issue the **ROLLFORWARD DATABASE...STOP** command to complete the roll-forward operation at the last consistent point just before the missing log file.

Although missing log detection ensures that database corruption does not occur as a result of missing log files, the presence of missing log files prevents some transactions from being replayed and, as a result, data loss could occur if the missing log files are not located.

## Required resources

Log stream merge operations require additional EDUs. During database activation, one **db21fr** EDU is created on each member. When a log read operation that requires a log stream merge is initiated, one **db2shred** EDU and one **db21fr** EDU is created for each log stream. Although each **db21fr-db2shred** group allocates its own set of log page and log record buffers, this is not a significant amount of additional memory or system resources; approximately 400 KB is allocated for each member that is involved in the log stream merge.

During a log stream merge operation, a member retrieves log files that are owned by other members into its overflow log path, primary log path, or mirror log path. In a Db2 pureScale environment, ensure that there is adequate free disk space in the retrieval path before starting a roll-forward operation. This allows the operation to retrieve the larger number of files from the archive, as required in a Db2 pureScale environment, without affecting performance. Use the following rule-of-thumb to calculate how much space you need to retrieve the active log files for all members: **(logprimary + logsecond) \* number of members**.

## Examples

- Update the **newlogpath** global database configuration parameter:  
db2 update db cfg for db mydb using newlogpath /home/dbuser/logdir
- Update the **max\_log** per-member database configuration parameter on a single member:  
db2 update db cfg for db mydb member 1 using max\_log 5
- Update the primary log path:  
db2 connect to mydb  
db2 update db cfg for mydb using newlogpath /home/dbuser/newlogpath  
db2 get db cfg for mydb  
...  
Changed path to log files (NEWLOGPATH) = /home/dbuser/newlogpath/NODE0000/LOGSTREAM0000/  
Path to log files = /home/dbuser/dbuser/NODE0000/LOGSTREAM0000/  
...

The change does not take effect because the member is still active.

```
db2 terminate
db2 deactivate db mydb
db2 connect to mydb
db2 get db cfg for mydb
...
Changed path to log files (NEWLOGPATH) =
Path to log files                       = /home/dbuser/newlogpath/NODE0000/LOGSTREAM0000/
...
```

Each member uses the /home/dbuser/newlogpath/NODE0000/LOGSTREAMxxxx log path, where xxxx is the log stream ID of the log stream that uses the path.

- Set a new primary log path while restoring a backup image:  
db2 restore db mydb newlogpath '/home/dbuser/newlogpath' without prompting

## Log sequence numbers in Db2 pureScale environments

Db2 databases use the log sequence number (LSN), a 64-bit identifier, to determine the order of the operations that generated the log records.

The LSN is an ever-increasing value. Each member writes to its own set of log files (a *log stream*), and the LSN within a single log stream is a unique number.

Because LSNs are generated independently on each member and there are multiple log streams, it is possible to have duplicate LSN values across different log streams. A log record identifier (LRI) is used to identify log records across log streams; each log record in any log stream in the database is assigned a unique LRI. Use the **db2pd** command to determine which LRI is being processed by a recovery operation.

---

## Recovery history file

A recovery history file is created with each database and is automatically updated during various operations.

The following operations cause the recovery history file to be updated:

- A database or table spaces are backed up
- A database or table spaces are restored
- A database or table spaces are rolled forward
- A database is automatically rebuilt and more than one image is restored
- A table space is created
- A table space is altered
- A table space is quiesced
- A table space is renamed
- A table space is dropped
- A table is loaded
- A table is dropped (when dropped table recovery is enabled and you are using recoverable logging)
- A table is reorganized
- On-demand log archiving is invoked
- A new log file is written to (when using recoverable logging)
- A log file is archived (when using recoverable logging)
- A database is recovered

- A failed restore database or table space operation

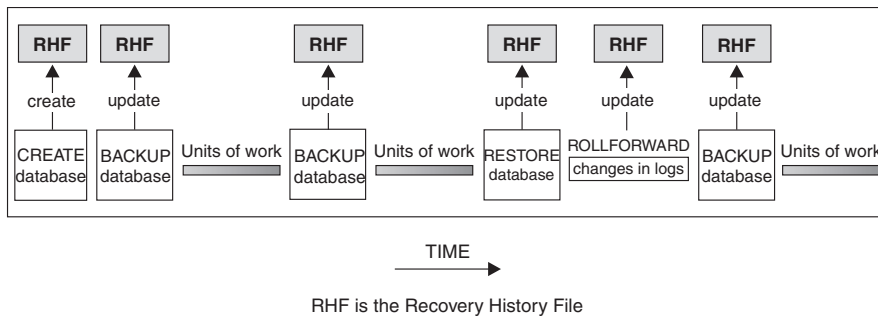


Figure 12. Creating and updating the recovery history file

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

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

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

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

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



## Recovery history file entry status

The database manager creates entries in the recovery history file for events such as a backup operation, a restore operation, table space creation, and others. Each entry in the recovery history file has an associated status: active, inactive, expired, pending delete, deleted, or do\_not\_delete.

The database manager uses the status of a recovery history file entry to determine whether the physical files associated with that entry would be needed to recover the database. As part of automated pruning, the database manager updates the status of recovery history file entries.

### Active database backup

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.

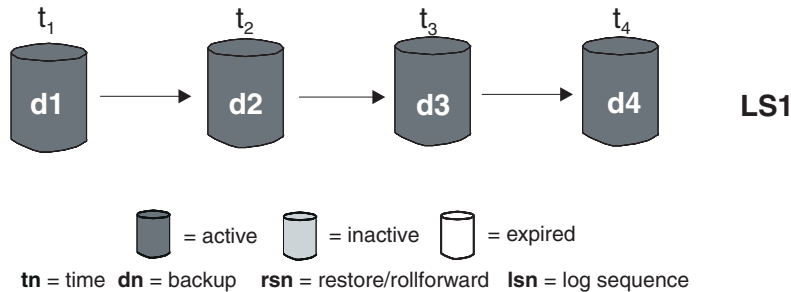


Figure 13. Active Database Backups. The value of num\_db\_backups has been set to four.

### Inactive database backup

An inactive database backup is one that, if restored, moves the database back to a previous state.

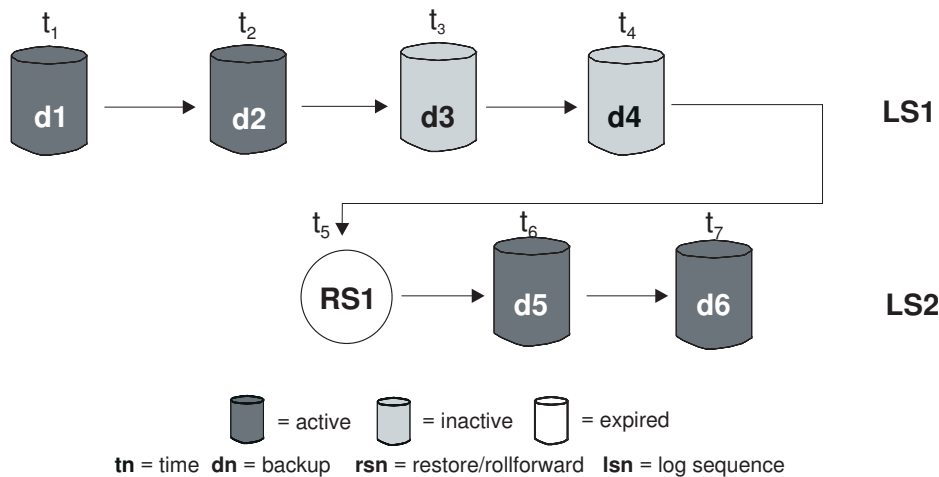


Figure 14. Inactive Database Backups

## Expired database backups

An expired database backup image is one that is no longer needed, because more recent backup images are available.

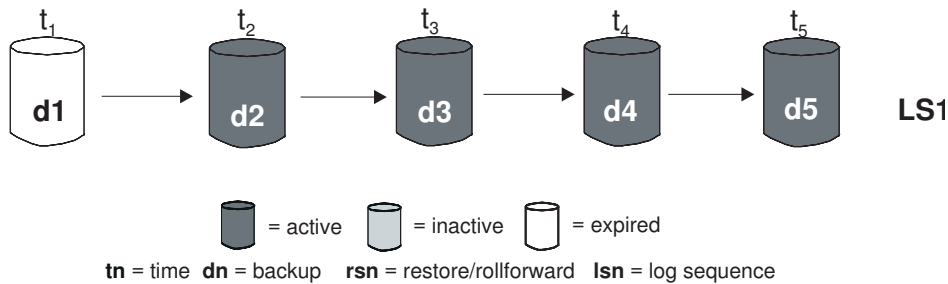


Figure 15. Expired Database Backups

## Entries marked do\_not\_delete

You can remove (prune) recovery history file entries using the **PRUNE HISTORY** command or the db2Prune API. The database manager also prunes the recovery history file entries as part of automated pruning.

There are only three ways to prune entries marked do\_not\_delete:

- Invoke the **PRUNE HISTORY** command with the **WITH FORCE** option
- Call the ADMIN\_CMD procedure with **PRUNE HISTORY** and **WITH FORCE** option
- Call the db2Prune API with the DB2\_PRUNE\_OPTION\_FORCE option

Those entries that are marked do\_not\_delete will never be pruned from the recovery history file unless you perform one of these three actions.

The database manager does not set the status of recovery history file entries to do\_not\_delete. You can set the status of a recovery history file entry to do\_not\_delete using the **UPDATE HISTORY** command.

## Entries marked delete\_pending

An entry marked pending\_delete is in the process of being removed. It may remain if a prune operation was terminated part way through. In this case, the file associated with the entry may or may not still exist, and is treated as if it did not exist (as with deleted entries).

Here are more examples of the status of different recovery history file entries:

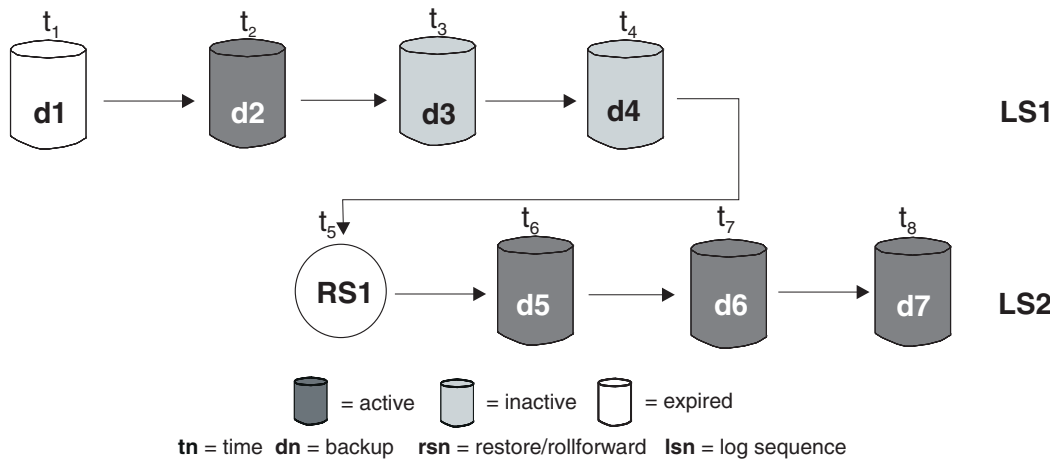


Figure 16. Mixed Active, Inactive, and Expired Database Backups

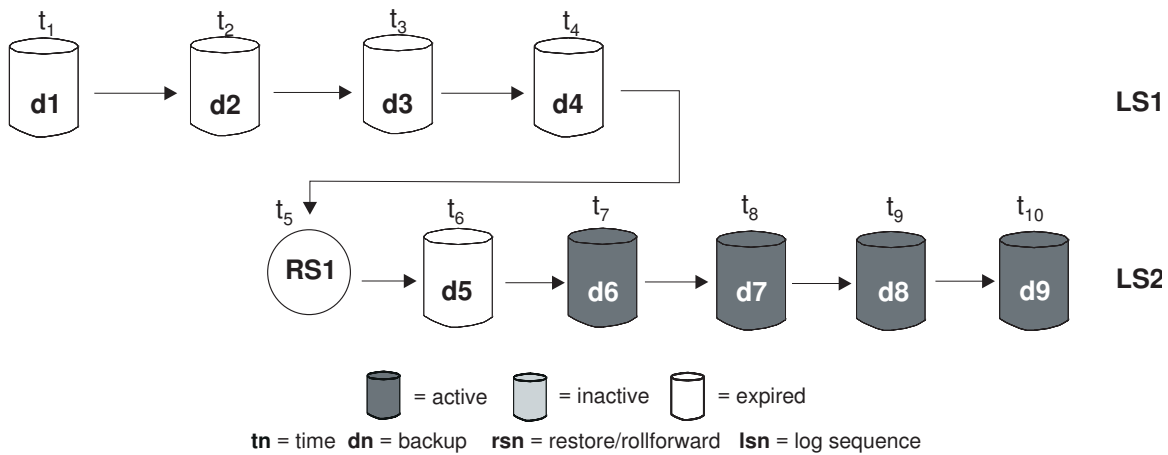


Figure 17. Expired Log Sequence

## Viewing recovery history file entries using the DB\_HISTORY administrative view

You can use the DB\_HISTORY() administrative view to access the contents of the database history file. This method is an alternative to using the **LIST HISTORY** CLP command or the C history APIs.

### Before you begin

A database connection is required to use this function.

### About this task

Deletes and updates to the database history file can be done only through the **PRUNE HISTORY** or **UPDATE HISTORY** commands.

## Procedure

Use the DB\_HISTORY() administrative view within an SQL SELECT statement to access the database history file for the database you are connected to, or on the database partition specified by the **DB2NODE** environment variable. For example, to see the contents of the history file use:

```
SELECT * FROM TABLE(DB_HISTORY()) AS LIST_HISTORY
```

## Example

To hide the syntax of the administrative view, you can create a view as follows:

```
CREATE VIEW LIST_HISTORY AS
SELECT * FROM TABLE(DB_HISTORY()) AS LIST_HISTORY
```

After creating this view, you can run queries against the view. For example:

```
SELECT * FROM LIST_HISTORY
```

or

```
SELECT dbpartitionnum FROM LIST_HISTORY
```

or

```
SELECT dbpartitionnum, start_time, seqnum, tabname, sqlstate
FROM LIST_HISTORY
```

For a list of columns and column data types returned by the DB\_HISTORY administrative view, see DB\_HISTORY administrative view.

## Pruning the recovery history file

The database manager creates entries in the recovery history file for events such as a backup operation, a restore operation, table space creation, and others.

When an entry in the recovery history file is no longer relevant, because the associated recovery objects would no longer be needed to recover the database, you might want to remove, or *prune*, those entries from the recovery history file.

## Procedure

You can prune the entries in the recovery history file using the following methods:

- Invoke the **PRUNE HISTORY** command
- Call the db2Prune API
- Call the ADMIN\_CMD procedure with the PRUNE\_HISTORY parameter

## What to do next

When you use one of these methods to prune the recovery history file, the database manager removes (prunes) entries from the recovery history file that are older than a timestamp you specify.

If a recovery history file entry matches the criteria you specify for pruning, but that entry would still be needed for a recovery of the database, the database manager will not prune the entry unless you use the **WITH FORCE** parameter or the DB2PRUNE\_OPTION\_FORCE flag.

If you use the **AND DELETE** parameter or the `DB2PRUNE_OPTION_DELETE` flag, then log files associated with pruned entries will be deleted as well.

If you set the **AUTO\_DEL\_REC\_OBJ** database configuration parameter to ON, and you use the **AND DELETE** parameter or the `DB2PRUNE_OPTION_DELETE` flag, then log files, backup images, and load copy images associated with pruned entries will be deleted.

## Automating recovery history file pruning

You can configure the database manager to automatically prune and update the status of recovery history file entries.

You can manually update the status of entries in the recovery history file using the **UPDATE HISTORY** command, the `db2HistoryUpdate` API, or the `ADMIN_CMD` procedure with the "UPDATE\_HISTORY" parameter. You can use the **PRUNE HISTORY** command, the `db2Prune` API, or the `ADMIN_CMD` procedure with the "PRUNE\_HISTORY" parameter to manually remove, or prune, entries from the recovery history file. However, it is recommended that you configure the database manager to manage the recovery history file instead of updating and pruning the recovery history file manually.

The database manager automatically updates and prunes recovery history file entries at the following times:

- After a full database backup operation or full table space backup operation completes successfully
- After a database restore operation, where a rollforward operation is not required, completes successfully
- After a database rollforward operation completes successfully

During automated pruning, the database manager performs two operations:

1. Updates the status of the recovery history file entries
2. Prunes expired recovery history file entries

The database manager updates the recovery history file entries in the following way:

- All active database backup images that are no longer needed are marked as expired.
- All database backup images that are marked as inactive and that were taken before 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 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.
- Any database or table space backup image that does not correspond to the current log sequence, also called the current log chain, is marked inactive.

The current log sequence is determined by the 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.
- Any entries that have a status of `do_not_delete` are not pruned, and their associated log files, backup images, and load copy images are not deleted.
- When a database is upgraded, all online database backup entries and all online or offline table space backup entries in the history file are marked as expired, so that these entries are not selected by automatic rebuild as images required for rebuilding. Load copy images and log archive entries are also marked as expired, since these types of entries cannot be used for recovery purposes

The following database configuration parameters control which entries the database manager prunes:

**num\_db\_backups**

Specifies the number of database backups to retain for a database.

**rec\_his\_retentn**

Specifies the number of days that historical information about backups is retained.

**auto\_del\_rec\_obj**

Specifies whether the database manager deletes log files, backup images, and load copy images that are associated with recovery history file entries that are pruned.

To configure the database manager to automatically manage the recovery history file, set the following configuration parameters:

- **num\_db\_backups**
- **rec\_his\_retentn**
- **auto\_del\_rec\_obj**

When **auto\_del\_rec\_obj** is set to ON, and whenever there are more successful database backup entries than the **num\_db\_backups** configuration parameter, then the database manager automatically prunes recovery history file entries that are older than **rec\_his\_retentn**.

## Protecting recovery history file entries from being pruned

You can prevent key recovery history file entries from being pruned, and associated recovery objects from being deleted by setting the status of the recovery history files entries to `do_not_delete`.

### About this task

You can remove (prune) recovery history file entries using the **PRUNE HISTORY** command, the ADMIN\_CMD procedure with **PRUNE\_HISTORY**, or the db2Prune API. If you use the **AND DELETE** parameter with **PRUNE HISTORY**, or the

DB2PRUNE\_OPTION\_DELETE flag with db2Prune, and the **auto\_del\_rec\_obj** database configuration parameter is set to ON, then the associated recovery objects will also be physically deleted.

The database manager also prunes the recovery history file entries as part of automated pruning. If the **auto\_del\_rec\_obj** database configuration parameter is set to ON, the database manager will delete the recovery objects associated with any entries that are pruned.

## Procedure

To protect key recovery history file entries and associated recovery objects:

Use the **UPDATE HISTORY** command, the db2HistoryUpdate API, or the ADMIN\_CMD procedure with "UPDATE\_HISTORY" to set the status for key recovery file entries to do\_no\_delete.

There are three ways to prune entries marked do\_not\_delete:

- Invoke the **PRUNE HISTORY** command with the **WITH FORCE** option
- Call the ADMIN\_CMD procedure with **PRUNE HISTORY** and **WITH FORCE** option
- Call the db2Prune API with the DB2\_PRUNE\_OPTION\_FORCE **ioption**.

Those entries that are marked do\_not\_delete will never be pruned from the recovery history file unless you perform one of these three procedures.

### Restrictions:

- You can set the status of only backup images, load copy images, and log files to do\_not\_delete.
- The status of a backup entry is not propagated to load copy images, non-incremental backups, or log files related to that backup operation. If you want to save a particular database backup entry and its related log file entries, you must set the status for the database backup entry and the entry for each related log file.

---

## Managing recovery objects

As you regularly backup your database, you might accumulate very large database backup images, and many database logs and load copy images. The IBM Data Server database manager can simplify managing these recovery objects.

### About this task

Storing recovery objects can consume great amounts of storage space. Once subsequent backup operations are run, you can delete the older recovery objects because they are no longer needed to restore the database. However, removing the older recovery objects can be time consuming. Also, while you are deleting the older recovery objects, you might accidentally damage recovery objects that are still needed.

## Procedure

There are two ways to use the database manager to delete recovery objects that are no longer required to restore the database:

- You can invoke the **PRUNE HISTORY** command with the **AND DELETE** parameter, or call the db2Prune API with the DB2PRUNE\_OPTION\_DELETE flag.
- You can configure the database manager to automatically delete unneeded recovery objects.

## Deleting database recovery objects using the PRUNE HISTORY command or the db2Prune API

You can use the **auto\_del\_rec\_obj** database configuration parameter and the **PRUNE HISTORY** command or the db2Prune API to delete recovery objects.

### About this task

When you invoke the **PRUNE HISTORY** command, or call the db2Prune API, the IBM Data Server database manager does the following:

- Prunes entries from the recovery history file that do not have the status **DB2HISTORY\_STATUS\_DO\_NOT\_DEL**

When you invoke the **PRUNE HISTORY** command with the **AND DELETE** parameter, or when you call the db2Prune API with the **DB2PRUNE\_OPTION\_DELETE** flag, the database manager does the following:

- Prunes entries from the recovery history file that are older than a timestamp you specify and that do not have the status **DB2HISTORY\_STATUS\_DO\_NOT\_DEL**
- Deletes the physical log files associated with the pruned entries

If you set the **auto\_del\_rec\_obj** database configuration parameter to **ON**, then when you invoke the **PRUNE HISTORY** command with the **AND DELETE** parameter, or when you call the db2Prune API with the **DB2PRUNE\_OPTION\_DELETE** flag, the database manager does the following:

- Prunes entries from the recovery history file that do not have the status **DB2HISTORY\_STATUS\_DO\_NOT\_DEL**
- Deletes the physical log files associated with the pruned entries
- Deletes the backup images associated with the pruned entries
- Deletes the load copy images associated with the pruned entries

### Procedure

To delete unneeded recovery objects:

1. Set the **auto\_del\_rec\_obj** database configuration parameter to **ON**.
2. Invoke the **PRUNE HISTORY** command with the **AND DELETE** parameter, or call the db2Prune API with the **DB2PRUNE\_OPTION\_DELETE** flag.

## Automating database recovery object management

You can use the **auto\_del\_rec\_obj** database configuration parameter and automated recovery history file pruning to configure the IBM Data Server database manager to automatically delete unneeded recovery objects after every full database backup operation.

### About this task

After every successful full database backup operation or full table space backup operation, the database manager prunes the recovery history file according to the settings of the **num\_db\_backup** and **rec\_his\_retentn** configuration parameters when all of the following conditions are true:

- There are more database backup entries in the recovery history file than the value of the **num\_db\_backups** configuration parameter.
- The database backup entries do not have their status set to **do\_not\_delete**.



- The database backup entries from the recovery history file are older than the value specified by the **rec\_his\_retentn** configuration parameter.

**Note:** If the **rec\_his\_retentn** configuration parameter is set to 0, then the automatic pruning is based on the setting for the **num\_db\_backups** parameter.

If you set the **auto\_del\_rec\_obj** database configuration parameter to ON, then the database manager will do the following in addition to pruning entries from the recovery history file:

- Delete the physical log files associated with the pruned entries
- Delete the backup images associated with the pruned entries
- Delete the load copy images associated with the pruned entries

If there are no full database backup images available for consideration in the current recovery history (perhaps none were ever taken), then images older than the range of time specified by **rec\_his\_retentn** will be deleted.

If the database manager is unable to delete a file because the file is no longer at the location listed in the recovery history file, then the database manager will prune the history entry.

If the database manager is unable to delete a file because of a communication error between the database manager and the storage manager or device, then the database manager will not prune the history file entry. When the error is resolved, the file can be deleted during the next automated prune.

## Procedure

To configure the database manager to automatically delete unneeded recovery objects:

1. Set the **auto\_del\_rec\_obj** database configuration parameter to ON.
2. Set the **rec\_his\_retentn** and **num\_db\_backups** configuration parameters to enable automated recovery history file pruning.

## Example

Consider the following scenario, which shows how the settings for automatic deletion interact. User1's backup plan specifies a weekly full database backup, with two incremental backups during the week. User1 has the following configuration:

- **auto\_del\_rec\_obj**=ON
- **rec\_his\_retentn**=0
- **num\_db\_backups**=3

In this scenario, User1 keeps three weeks of history, three full backups, and all of the incremental backups and logs in between those backups. With this configuration, if User1 changes to daily backups, User1 keeps three days of history, three full backups, and all of the incremental backups and logs in between those backups.

If User1 changes to the following configuration:

- **auto\_del\_rec\_obj**=ON
- **rec\_his\_retentn**=15
- **num\_db\_backups**=3

User1 still keeps three weeks of history, three full backups, and all of the incremental backups and logs in between those backups. However, if User1 changes to daily backups, User1 keeps 15 days of history, 15 full backups, and all of the incremental backups and logs in between those backups.

## Protecting recovery objects from being deleted

Automated recovery object management saves administration time and storage space. However, you might want to prevent certain recovery objects from being automatically deleted. You can prevent key recovery objects from being deleted by setting the status of the associated recovery history files entries to `do_not_delete`.

### About this task

If you set the **auto\_del\_rec\_obj** database configuration parameter to ON, then recovery objects get deleted when their associated recovery history file entries get pruned. Recovery history file entries get pruned when one of the following happens:

- You invoke the **PRUNE HISTORY** command with the **AND DELETE** parameter
- You call the `db2Prune` API with the `DB2PRUNE_OPTION_DELETE` flag
- The database manager automatically prunes the recovery history file, which happens after every successful table space or database full backup.

Whether you invoke the **PRUNE HISTORY** command, call the `db2Prune` API, or configure the database manager to automatically prune the entries in the recovery history file, entries that are marked `do_not_delete` are not pruned, and the associated recovery objects are not deleted.

#### Restrictions

- You can set the status of only backup images, load copy images, and log files to `do_not_delete`.
- The status of a backup entry is not propagated to log files, load copy images, or non-incremental backups related to that backup operation. If you want to save a particular database backup entry and its related log file entries, you must set the status for the database backup entry and the entry for each related log file.

### Procedure

Use the **UPDATE HISTORY** command to set the status for associated recovery file entries to `do_no_delete`.

## Managing snapshot backup objects

You must use the **db2acsutil** command to manage snapshot backup objects. Do not move or delete snapshot backup objects using file system utilities.

### Before you begin

To perform snapshot backup and restore operations, you need one of two things:

- A Db2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.
- For storage devices that are not supported, implement a custom script that allows your storage device to perform snapshot operations.

#### Restrictions

The **db2acsutil** command is currently only supported on AIX and Linux.

## Procedure

- To list available snapshot backup objects, use the **QUERY** parameter.  
For example, to list available snapshot backup objects for a specific database manager instance, use the following syntax:  
`db2acsutil query instance inst-name`
- To check the progress of a snapshot backup operation, use the **STATUS** parameter.  
For more verbose progress information, use the **SHOW DETAILS** parameter as well.  
For example, to see the progress of snapshot backup operations that might be currently running on a specific database, use the following syntax:  
`db2acsutil query status db db-name`
- To delete a particular snapshot backup object, use the **DELETE** parameter.  
For example, to delete all snapshot backup objects for a given database older than a specific period of time, use the following syntax:  
`db2acsutil delete older than number days ago db db-name`

**Note:** For each of these tasks, you can also query or delete a specific set of snapshot objects by their origin by using one of the following options:

- The **LOAD** and **OPTION** parameter to specify the shared library that contains the vendor fast copying technology used for snapshot backup, as in this example:  
`db2acsutil load path-to-library  
options 'option1 optionN'  
...`
- The **SCRIPT** parameter to specify the snapshot backup objects created by a custom script, as in this example:  
`db2acsutil script "path-to-script" ...`

–

## Backup image and log file upload to IBM Tivoli Storage Manager (TSM)

You can choose to back up to disk first in a relatively shorter time and later upload the backup image and log files to Tivoli Storage Manager (TSM) while maintaining the recovery history information so it appears as if they were backed up directly to TSM.

This strategy might be appropriate in situations where you are producing backup images faster than TSM can write them.

### Example 1: Adoption strategy

As a part of your recovery plan, you decide to keep a specific set of images and logs on disk to facilitate recovery, and at a predetermined interval—in this case, weekly—you upload the oldest images and logs to TSM. (Note that this scenario favors a fast recovery window and might not match everyone's requirements; some users, for example, would upload their backups to TSM immediately.) The procedure would be to query the recovery history file for the oldest backup image, and then to upload that image and its logs to TSM.

1. Query the history file for available logs and images using the following command:  
`db2 list history all for db sample`

The following information is returned:

List History File for sample

Number of matching file entries = 100

...  
...  
...

Op	Obj	Timestamp+Sequence	Type	Dev	Earliest Log	Current Log	Backup ID
X	D	20110403134938	1	D	S0000003.LOG		C0000000

Comment:

Start Time: 20110403134938

End Time: 20110403135204

Status: A

EID: 5 Location: /home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000003.LOG

...  
...  
...

Op	Obj	Timestamp+Sequence	Type	Dev	Earliest Log	Current Log	Backup ID
B	D	20110404135750001	F	D	S0000000.LOG	S0000007.LOG	

Contains 2 tablespace(s):

00001 SYSCATSPACE

00002 USERSPACE1

Comment: DB2 BACKUP SAMPLE OFFLINE

Start Time: 20110404135750

End Time: 20110404135755

Status: A

EID: 10 Location: /home/backupdir

...  
...  
...

2. You choose the oldest log file to upload using the following command:  
db2adutl upload logs between s3 and s3 db sample

The following information is returned:

```
=====
| Upload Summary: |
=====
```

1 / 1 logs were successfully uploaded

Logs successfully uploaded:

/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000003.LOG

3. You upload the oldest image and its logs using the following command:  
db2adutl upload images taken at 20110404135750 with logs db sample

The following information is returned:

Match found, but S0000003.LOG is already on TSM

```
=====
| Upload Summary: |
=====
```

1 / 1 backup images were successfully uploaded

4 / 4 logs were successfully uploaded

Backup Images successfully uploaded:

```
/home/backupdir/SAMPLE.0.dwu.NODE0000.CATN0000.20110404135750.001
```

```
Logs successfully uploaded:
/home/logdir/log1/dwu/SAMPLE/NODE0000/C0000001/S0000004.LOG
/home/logdir/log1/dwu/SAMPLE/NODE0000/C0000001/S0000005.LOG
/home/logdir/log1/dwu/SAMPLE/NODE0000/C0000002/S0000006.LOG
/home/logdir/log1/dwu/SAMPLE/NODE0000/C0000002/S0000007.LOG
```

4. You verify the results:

- a. by querying the history file using the following command:
- ```
db2 connect to sample
```

The following information is returned:

```
Database server = DB2/LINUX8664 9.7.5
SQL authorization ID = DWU
Local database alias = SAMPLE
```

```
db2 select OPERATION, OBJECTTYPE, START_TIME, SEQNUM, FIRSTLOG, LASTLOG, LOCATION,
DEVICETYPE from table(DB_HISTORY()) as T
```

The following information is returned:

| OPERATION | OBJECTTYPE | START_TIME     | SEQNUM | FIRSTLOG     | LASTLOG  | LOCATION      | DEVICETYPE |
|-----------|------------|----------------|--------|--------------|----------|---------------|------------|
| X         | D          | 20110403134938 | -      | S0000003.LOG | C0000000 | adsm/libtsm.a | A          |
| ...       |            |                |        |              |          |               |            |
| B         | D          | 20110404135750 | 1      | S0000000.LOG | S0000007 | adsm/libtsm.a | A          |

- b. by querying TSM using the following command:
- ```
db2adutl query db sample
```

The following information is returned:

Query for database SAMPLE

```
Retrieving FULL DATABASE BACKUP information.
1 Time: 20110404135750 Oldest log: S0000007.LOG DB Partition Number: 0 Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for SAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.
No DELTA DATABASE BACKUP images found for SAMPLE
```

```
Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for SAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for SAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for SAMPLE
```

```
Retrieving LOAD COPY information.
No LOAD COPY images found for SAMPLE
```

```
Retrieving LOG ARCHIVE information.
Log file: S0000003.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.28
Log file: S0000004.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.29
Log file: S0000005.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.30
Log file: S0000006.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.30
Log file: S0000007.LOG, Chain Num: 1, DB Partition Number: 0, Taken at: 2011-04-04-21.38.31
```

5. The next week, you upload the oldest backup image using the following command:

```
db2adutl upload images taken at 20110409155645 with logs db sample
```

The following information is returned:

```
=====
| Upload Summary: |
=====
```

```
1 / 1 backup images were successfully uploaded
2 / 2 logs were successfully uploaded
```

```
Backup Images successfully uploaded:
/home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110409155645.001
```

```
Logs successfully uploaded:
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000008.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000009.LOG
```

6. You verify the results by querying TSM using the following command:

```
db2adutl query db sample
```

The following information is returned:

Query for database SAMPLE

```
Retrieving FULL DATABASE BACKUP information.
1 Time: 20110404135750 Oldest log: S0000007.LOG DB Partition Number: 0 Sessions: 1
2 Time: 20110409155645 Oldest log: S0000009.LOG DB Partition Number: 0 Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for SAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.
No DELTA DATABASE BACKUP images found for SAMPLE
```

```
Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for SAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for SAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for SAMPLE
```

```
Retrieving LOAD COPY information.
No LOAD COPY images found for SAMPLE
```

```
Retrieving LOG ARCHIVE information.
Log file: S0000003.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.28
Log file: S0000004.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.29
Log file: S0000005.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.30
Log file: S0000006.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.30
Log file: S0000007.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-04-21.38.31
Log file: S0000008.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-09-20.21.50
Log file: S0000009.LOG, Chain Num: 0, DB Partition Number: 0, Taken at: 2011-04-09-20.21.51
```

## Example 2: Uploading and removing a local backup image

1. You take a backup of your database as follows:

```
db2 backup db sample to /home/backupdir
```

The following information is returned:

Backup successful. The timestamp for this backup image is: 20110401135620

2. At a later time, you decide to upload that backup image and erase it from disk, using the following command:

```
db2adutl upload and remove images taken at 20110401135620 db sample
```

The following information is returned:

```
File /home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110401135620.001 is uploaded successfully.
Would you really like to remove the original file (Y/N)
```

3. You enter Y.

**Note:** If you wanted to perform the upload without being prompted before removing the backup image from disk you would use the following command:

```
db2adutl upload and remove images taken at 20110401135620 db sample without prompting
```

The following information is returned:

```
/home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110401135620.001 is successfully removed.
```

```
=====
| Upload Summary: |
=====
```

```
1 / 1 backup images were successfully uploaded
```

```
Backup Images successfully uploaded:
/home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110401135620.001
```

### Example 3: Uploading an image with no timestamp

1. You upload a backup image without specifying a timestamp or file name using the following command:

```
db2adutl upload images db sample
```

2. You are prompted about whether or not you want to upload the most recent image:

```
Upload the most recent backup image?
```

3. You enter Y.

The following information is returned:

```
=====
| Upload Summary: |
=====
1 / 1 backup images were successfully uploaded
Backup Images successfully uploaded:
/home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110401160128.001
```

If the most recent backup image already exists on TSM, the following information would be returned:

```
The most recent image is already on TSM.
```

### Example 4: Uploading a logs and a specific image

You want to upload a specific backup image and to include its logs, so you issue the following command:

```
db2adutl upload images taken at 20110401155645 with logs db sample
```

The following information is returned:

```

=====
| Upload Summary: |
=====

1 / 1 backup images were successfully uploaded
5 / 5 logs were successfully uploaded

Backup Images successfully uploaded:
/home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110401155645.001

Logs successfully uploaded:
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000000.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000001.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000002.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000003.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000001/S0000004.LOG

```

If you wanted a specific set of logs to be uploaded with that image, you would specify the range of sequence numbers, as in the following command:

```
db2adutl upload images taken at 20110401155645 logs between s3 and s7 db sample
```

The following information is returned:

```

=====
| Upload Summary: |
=====

1 / 1 backup images were successfully uploaded
5 / 5 logs were successfully uploaded

Backup Images successfully uploaded:
/home/backupdir/SAMPLE.0.diwu.NODE0000.CATN0000.20110401155645.001

Logs successfully uploaded:
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000000/S0000003.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000001/S0000004.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000001/S0000005.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000002/S0000006.LOG
/home/logdir/log1/diwu/SAMPLE/NODE0000/C0000002/S0000007.LOG

```

---

## Backup overview

Create a backup of your Db2 database and related stored data to prevent data loss in the event of a database service outage. There are several tools that you can use to complete the backup process.

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
```

In IBM Data Studio Version 3.1 or later, you can use the task assistant for backing up databases. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see *Administering databases with task assistants*.

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. Backup images created by Db2 Version 9.5 and later are generated with file mode 600, meaning that on



UNIX only the instance owner has read and write privileges and on Windows only members of the DB2ADMNS (and Administrators) group have access to the backup images.

**Note:** If the Db2 client and server are not located on the same system, Db2 database systems will determine which directory is the current working directory on the client machine and use that as the backup target directory on the server. For this reason, it is recommended that you specify a target directory for the backup image.

Backup images are created at the target location specified when you invoke the backup utility. This location can be:

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

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

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

If your recovery objects were saved using Tivoli Storage Manager (TSM), you can use the **db2adutl** utility to query, extract, verify, and delete the recovery objects. On Linux and UNIX, this utility is located in the `sqlllib/adsm` directory, and on Windows operating systems, it is located in `sqlllib\bin`. For snapshots, use the **db2acsutil** utility located in `sqlllib/bin`.

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

`DB_alias.Type.Inst_name.DBPARTnnn.timestamp.Seq_num`

For example:

`STAFF.0.DB201.DBPART000.19950922120112.001`

**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.

**Database partition number**

In single partition database environments, this is always `DBPART000`. In partitioned database environments, it is `DBPARTxxx`, where `xxx` is the number assigned to the database partition in the `db2nodes.cfg` file.

**Time stamp**

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

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

**Sequence number**

A 3-digit number used as a file extension.

When a backup image is written to tape:

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

You cannot back up a database that is not in a normal or backup-pending state. A table space that is in a normal or backup-pending state can be backed up. If the table space is not in a normal or backup-pending state, a backup may or may not be permitted.

Concurrent backup operations on the same table space are not permitted. Once a backup operation has been initiated on a table space, any subsequent attempts will fail (SQL2048N).

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

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

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

## Displaying backup information

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

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

## Backing up data

Use the **BACKUP DATABASE** command to take a copy of the database data and store it on a different medium. This backup data can then be used in the case of a failure or damage to the original data.

You can back up an entire database, database partition, or only selected table spaces.

### Before you begin

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

The database can be local or remote. The backup image remains on the database server, unless you are using a storage management product such as Tivoli Storage Manager (TSM) or Db2 Advanced Copy Services (ACS).

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

```
CONNECT TO database-alias
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS;
UNQUIESCE DATABASE;
TERMINATE;
DEACTIVATE DATABASE database-alias
```

In a partitioned database environment, you can use the **BACKUP DATABASE** command to back up database partitions individually, use the **ON DBPARTITIONNUM** command parameter to back up several of the database partitions at once, or use the **ALL DBPARTITIONNUMS** parameter to back up all of the database partitions simultaneously. You can use the **LIST DBPARTITIONNUMS** command to identify the database partitions that have user tables on them that you might want to back up.

Unless you are using a single system view (SSV) backup, if you are performing an *offline* backup in a partitioned database environment, you should back up the catalog partition separately from all other database partitions. For example, you can back up the catalog partition first, then back up the other database partitions. This action is necessary because the backup operation might require an exclusive database connection on the catalog partition, during which the other database

partitions cannot connect. If you are performing an *online* backup, all database partitions (including the catalog partition) can be backed up simultaneously or in any order.

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 upgraded, you must upgrade the database before you can back it up.

### Restrictions

The following restrictions apply to the backup utility:

- A table space backup operation and a table space restore operation cannot be run at the same time, even if different table spaces are involved.
- If you want to be able to do rollforward recovery in a partitioned database environment, you must regularly back up the database on the list of nodes. You must also have at least one backup image of the rest of the nodes in the system (even those nodes that do not contain user data for that database). Two situations require the backed-up image of a database partition at a database partition server that does not contain user data for the database:
  - You added a database partition server to the database system after taking the last backup, and you need to do forward recovery on this database partition server.
  - Point-in-time recovery is used, which requires that all database partitions in the system are in rollforward pending state.
- Online backup operations for DMS table spaces are incompatible with the following operations:
  - load
  - reorganization (online and offline)
  - drop table space
  - table truncation
  - index creation
  - not logged initially (used with the CREATE TABLE and ALTER TABLE statements)
- If you attempt to perform an offline backup of a database that is currently active, you will receive an error. Before you run an offline backup, you can make sure that the database is not active by issuing the **DEACTIVATE DATABASE** command.

### Procedure

To invoke the backup utility:

- Issue the **BACKUP DATABASE** command in the command line processor (CLP).
- Run the ADMIN\_CMD procedure with the BACKUP DATABASE parameter.
- Use the db2Backup application programming interface (API).
- Open the task assistant in IBM Data Studio for the **BACKUP DATABASE** command.

## Example

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

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

## What to do next

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

```
ACTIVATE DATABASE sample
```

## Performing a snapshot backup

A snapshot backup operation uses the fast copying technology of a storage device to perform the data copying portion of the backup.

## Before you begin

To perform snapshot backup and restore operations, you need one of two things:

- A Db2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.
- For storage devices that are not supported, implement a custom script that allows your storage device to perform snapshot operations.

Before you can perform a snapshot backup, you must enable Db2 Advanced Copy Services (ACS). See: “Enabling Db2 Advanced Copy Services (ACS)” on page 442.

### Restrictions

You cannot recover individual table spaces by using snapshot backups.

If you use integrated snapshot backups, you cannot perform a redirected restore. A FlashCopy® restore reverts the complete set of volume groups containing all database paths to a prior point in time.

## Procedure

To perform a snapshot backup, use one of the following approaches:

- Issue the **BACKUP DATABASE** command with the **USE SNAPSHOT** parameter, as shown in the following example:

```
db2 backup db sample use snapshot
```

- Call the ADMIN\_CMD procedure with **BACKUP DB** and **USE SNAPSHOT** parameters, as shown in the following example:

```
CALL SYSPROC.ADMIN_CMD  
('backup db sample use snapshot')
```

- Issue the db2Backup API with the SQLU\_SNAPSHOT\_MEDIA media type, as shown in the following example:

```
int sampleBackupFunction( char dbAlias[],  
                          char user[],  
                          char pswd[],  
                          char workingPath[] )  
{  
    db2MediaListStruct mediaListStruct = { 0 };  
  
    mediaListStruct.locations = &workingPath;
```

```

mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;

db2BackupStruct backupStruct = { 0 };

backupStruct.piDBAlias = dbAlias;
backupStruct.piUsername = user;
backupStruct.piPassword = pswd;
backupStruct.piVendorOptions = NULL;
backupStruct.piMediaList = &mediaListStruct;

db2Backup(db2Version950, &backupStruct, &sqlca);

return 0;
}

```

### Performing a snapshot backup with a script:

Using a custom script allows you to perform snapshot backup operations to storage devices that are not supported by Db2 ACS

#### Before you begin

You must have one of the following authorities: SYSADM, SYSCTRL, or SYSMAINT.

#### About this task

Snapshot backups allow you to use the functionality of your underlying storage system to instantly create a copy of all database data and transaction logs without any interruptions. With a custom script, you can specify various options for the snapshot backup operation as well as utilize a wide range of storage devices that do not provide a vendor library.

During online snapshot backups, the database manager temporarily suspends all write operations to disk before creating the snapshot. This ensures that no changes occur to the data during the few seconds when the snapshot is taken.

#### Procedure

To perform a snapshot backup:

1. Create a script that implements the Db2 ACS API. The script must be executable. For information on custom scripts, see “Db2 Advanced Copy Services (ACS) user scripts” on page 457.
2. Optional: Create a protocol file repository. This directory will contain the protocol files for the snapshot. Ensure that the directory is readable and writable.

If you do not create the repository, the protocol files will be written to the directory that contains your script.

3. Initiate<sup>®</sup> the backup operation using either the **BACKUP DATABASE** command, the ADMIN\_CMD procedure with BACKUP DB option, or the db2Backup API.

##### **BACKUP DATABASE command**

```

BACKUP DATABASE dbname ONLINE
USE SNAPSHOT SCRIPT path-to-script
OPTIONS 'path-to-repository additional options'

```

### ADMIN\_CMD procedure

```
CALL SYSPROC.ADMIN_CMD
  (backup database dbname online
   use snapshot script path-to-script
   options 'path-to-repository additional options')

```

### db2Backup API

```
int sampleBackupFunction( char dbAlias[],
                          char user[],
                          char pswd[],
                          char workingPath[] )
{
    db2MediaListStruct mediaListStruct = { 0 };

    mediaListStruct.locations = &workingPath;
    mediaListStruct.numLocations = 1;
    mediaListStruct.locationType = SQLU_SNAPSHOT_SCRIPT_MEDIA;

    db2BackupStruct backupStruct = { 0 };

    backupStruct.piDBAlias = dbAlias;
    backupStruct.piUsername = user;
    backupStruct.piPassword = pswd;
    backupStruct.piVendorOptions = NULL;
    backupStruct.piMediaList = &mediaListStruct;
    db2Backup(db2Version1050, &backupStruct, &sqlca);

    return 0;
}

```

## Results

The snapshot operation generates a snapshot backup image and a protocol file. Ensure that you keep the protocol file so it can be used for subsequent restore, query, delete operations.

## 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 database systems write out the backup image header as a 4-KB block. The only fixed block sizes Db2 database systems support are 512, 1024, 2048, and 4096 bytes. If you are using a fixed block size, you can specify any backup buffer size. However, you might find that your backup operation will not complete successfully if the fixed block size is not one of the sizes that Db2 database systems support.

If your database is large, using a fixed block size means that your backup operations might take more time than expected to complete. To improve performance, you can use a variable block size.

**Note:** When using a variable block size, ensure that you have well tested procedures in place that enable you to recover successfully, including explicitly specified buffer sizes for the **BACKUP** and **RESTORE** commands, with backup images that are created using a variable block size.

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

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

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

Where:

*device* is a valid tape device name. The default on Windows operating systems is \\.\TAPE0.

*blksize*

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

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

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

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

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

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

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

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

Device	Attachment	Block Size Limit	Db2 Buffer Size Limit (in 4-KB pages)
8 mm	scsi	131,072	32
3420	s370	65,536	16
3480	s370	61 440	15
3490	s370	61 440	15
3490E	s370	65,536	16
7332 (4 mm) <sup>1</sup>	scsi	262,144	64



Device	Attachment	Block Size Limit	Db2 Buffer Size Limit (in 4-KB pages)
3490e	scsi	262,144	64
3590 <sup>2</sup>	scsi	2,097,152	512
3570 (magstar MP)		262,144	64

**Note:**

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.

### Verifying the compatibility of your tape device

On UNIX, Linux, and AIX operating systems only, to determine whether your tape device is supported for backing up your Db2 databases, perform the following procedure:

As the database manager instance owner, run the operating system command **dd** to read from or write to your tape device. If the **dd** command succeeds, then you can back up your Db2 databases using your tape device.

### Backing up to named pipes

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

### Before you begin

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

### Procedure

1. Create a named pipe. The following is an AIX example:  

```
mkfifo /u/dmcinnis/mypipe
```
2. If this backup image is going to be used by the restore utility, the restore operation must be invoked *before* the backup operation, so that it does not miss any data:  

```
db2 restore db sample from /u/dmcinnis/mypipe into mynewdb
```
3. Use this pipe as the target for a database backup operation:  

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

## Backing up partitioned databases

Backing up a database in a partitioned database environment can pose difficulties such as tracking the success of the backup of each database partition, managing the multiple log files and backup images, and ensuring the log files and backup images for all the database partitions span the minimum recovery time that is required to restore the database.

Using a single system view (SSV) backup is the easiest way to back up a partitioned database.

## About this task

There are four ways to back up a database in a partitioned database environment:

- Back up each database partition one at a time by using the **BACKUP DATABASE** command, the **BACKUP DATABASE** command with the ADMIN\_CMD procedure, or the db2Backup API.
- Use the **db2\_a11** command with the **BACKUP DATABASE** command to first back up the catalog partition and then to back up a specified list of database partitions.
- Run a single system view (SSV) backup to back up some or all of the database partitions simultaneously, including the catalog partition.
- Use a task assistant in IBM Data Studio to guide you through the process of backing up the database.

Backing up each database partition one at a time is time-consuming and error-prone. Backing up all the partitions by using the **db2\_a11** command is easier than backing up each database partition individually because you generally only must make one command call. However, when you use **db2\_a11** to back up a partitioned database, you sometimes still must make multiple calls to **db2\_a11** because the database partition that contains the catalog cannot be backed up simultaneously with non-catalog database partitions. Whether you back up each database partition one at a time or use **db2\_a11**, managing backup images that were created using either of these methods is difficult because the time stamp for each database partition's backup image is different, and coordinating the minimum recovery time across the database partitions' backup images is difficult as well.

For the previously mentioned reasons, the recommended way to back up a database in a partitioned database environment is to use an SSV backup because you can decide to back up all database partitions simultaneously, including the catalog partition, and get the same time stamp for each database partition backup. Alternatively, you can split your backup, specifying some database partitions for which you get the same time stamp, and later take additional backups on the other database partitions to complete the database backup. The catalog partition can be backed up at any time with any other database partitions.

**Note:** For restore operations, you still must restore the catalog partition before you restore some or all of the other database partitions.

## Procedure

To back up some or all of the database partitions of a partitioned database simultaneously by using an SSV backup:

1. Optional: Allow the database to remain online, or take the database offline.  
You can back up a partitioned database while the database is online or offline. If the database is online, the backup utility acquires shared connections to the other database partitions, so user applications are able to connect to database partitions while they are being backed up.
2. On the database partition that contains the database catalog, perform the backup with appropriate parameters for partitioned databases, using one of the following methods:
  - Run the **BACKUP DATABASE** command with the **ON DBPARTITIONNUMS** parameter.

- Run the **BACKUP DATABASE** command with the **ON DBPARTITIONNUMS** parameter by using the ADMIN\_CMD procedure.
  - Call the db2Backup API with the **iAllNodeFlag** parameter.
  - Open the task assistant for the **BACKUP DATABASE** command in IBM Data Studio.
3. Optional: Include the log files that are required for recovery with the backup images.
- By default, log files are included with backup images if you are performing an SSV backup (that is, if you specify the **ON DBPARTITIONNUM** parameter). If you do not want log files to be included with the backup images, use the **EXCLUDE LOGS** command parameter when you run the backup. Log files are excluded from the backup image by default for non-SSV backups.
- For more information, see “Including log files with a backup image” on page 173.
4. Optional: Delete previous backup images. The method that you use to delete old backup images depends on how you store the backup images. For example, if you store the backup images to disk, you can delete the files; if you store the backup images using IBM Tivoli Storage Manager, you can use the **db2adutl** utility to delete the backup images. If you are using Db2 Advanced Copy Services (ACS), you can use the **db2acsutil** command to delete snapshot backup objects.

## Backing up partitioned tables using IBM Tivoli Space Manager Hierarchical Storage Management

The Tivoli Space Manager Hierarchical Storage Manager (HSM) client program automatically migrates eligible files to secondary storage to maintain specific levels of free space on local file systems.

With table partitioning, table data is divided across multiple storage objects called data partitions. HSM supports the backup of individual data partitions to secondary storage.

When using SMS table spaces, each data partition range is represented as a file in the corresponding directory. Therefore, it is very easy to migrate individual ranges of data (data partitions) to secondary storage.

When using DMS table spaces, each container is represented as a file. In this case, infrequently accessed ranges should be stored in their own table space. When you issue a CREATE TABLE statement using the EVERY clause, use the NO CYCLE clause to ensure that the number of table spaces listed in the table level IN clause match the number of data partitions being created. This is demonstrated in the following example:

### Example 1

```
CREATE TABLE t1 (c INT) IN tbsp1, tbsp2, tbsp3 NO CYCLE
PARTITION BY RANGE(c)
(STARTING FROM 2 ENDING AT 6 EVERY 2);
```

## Enabling automatic backup

A database can become unusable due to a wide variety of hardware or software failures. Ensuring that you have a recent, full backup of your database is an integral part of planning and implementing a disaster recovery strategy for your system.

Use automatic database backup as part of your disaster recovery strategy to enable Db2 to back up your database both properly and regularly.

## About this task

You can configure automatic backup using the command line interface, or the AUTOMAINT\_SET\_POLICY system stored procedure. You also need to enable the health indicator db.db\_backup\_req, which by default is enabled. Note that only an active database is considered for the evaluation.

## Procedure

- To configure automatic backup using the command line interface, set each of the following database configuration parameters to ON:
  - **AUTO\_MAINT**
  - **AUTO\_DB\_BACKUP**
- To configure automatic backup using IBM Data Studio, right-click the database and select the task assistant to configure automatic backup.
- To configure automatic backup using the AUTOMAINT\_SET\_POLICY system stored procedure:
  1. Create configuration XML input specifying details like backup media, whether the backup should be online or offline, and frequency of the backup. You can copy the contents of the sample file called DB2DefaultAutoBackupPolicy.xml in the SQLLIB/samples/automaintcfg directory and modify the XML to satisfy your configuration requirements.
  2. Optional: Create an XML input file containing your configuration XML input.
  3. Call AUTOMAINT\_SET\_POLICY with the following parameters:
    - maintenance type: AutoBackup
    - configuration XML input: either a BLOB containing your configuration XML input text; or the name of the file containing your configuration XML input.

See the topic “Configuring an automated maintenance policy using SYSPROC.AUTOMAINT\_SET\_POLICY or SYSPROC.AUTOMAINT\_SET\_POLICYFILE” on page 144 for more information about using the AUTOMAINT\_SET\_POLICY system stored procedure.

## Automatic database backup

A database might become unusable due to a variety of hardware or software failures. Automatic database backup simplifies database backup management tasks for the DBA by always ensuring that a recent full backup of the database is performed as needed.

It determines the need to perform a backup operation based on one or more of the following measures:

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

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

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

If backup to disk is selected, the automatic backup feature will regularly delete backup images from the directory specified in the automatic database backup configuration. Only the most recent backup image will be available at any given time, regardless of the number of full backups that are specified in the automatic backup policy file. It is recommended that this directory be kept exclusively for the automatic backup feature and not be used to store other backup images.

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

You can also configure automatic backup using one of the system stored procedures called **AUTOMAINT\_SET\_POLICY** and **AUTOMAINT\_SET\_POLICYFILE**.

## Backup and restore operations in a Db2 pureScale environment

In a Db2 pureScale environment, issuing a single **BACKUP DATABASE** or **RESTORE DATABASE** command on any member initiates a backup or restore operation on behalf of all members.

A backup operation generates a single backup image that includes data from the specified table spaces and any required metadata and configuration information for all currently defined members. You do not have to perform additional backup operations on any other member in the Db2 pureScale instance. Moreover, you require only a single **RESTORE DATABASE** command to restore the database and the member-specific metadata for all members. You do not have to perform additional restore operations on any other member to restore the cluster.

The time stamps of consecutive backup images are unique, increasing values, regardless of which member produced them.

You do not have to perform the restore operation on the same member that was used to generate the backup image, provided the backup image is accessible by the member performing the restore.

**Note:** If IBM Tivoli Storage Manager (TSM) is the target location on the **BACKUP DATABASE** command, it is recommended to configure the TSM client to use proxy node. This allows all members of the Db2 pureScale instance to access the backup image stored on the TSM server. See “Configuring a Tivoli Storage Manager client” on page 436 for details.

All members must be consistent before an offline backup operation can be attempted. Only one offline backup operation can run at one time, because the

backup utility acquires super-exclusive access to the database across all members. Although concurrent online backup operations are supported, different backup operations cannot copy the same table spaces simultaneously, and must wait their turn.

All of the reading of data and metadata from the database and all of the writing to a backup image takes place on a single member. Interactions between the backup or restore operation and other members are limited to copying or updating database metadata (such as table space definitions, the log file header, and the database configuration).

**Note:** Before taking a backup, you need to ensure that the log archiving path is set to a shared directory so that all the members are able to access the logs for subsequent rollforward operations. If the archive path is not accessible from the member on which the rollforward is being executed, SQL1273N is returned. The following command is an example of how to set the log path to the shared directory:

```
db2 update db cfg using logarchmeth1
      DISK:/db2fs/gpfs1/svtdbm5/svtdbm5/ArchiveLOGS
```

(where *gpfs1* is the shared directory for the members and *ArchiveLOGS* is the actual directory that archives the logs.

Online backup operations can proceed successfully if another member is offline, goes offline, or comes back online while the operation is executing (Table 19). Although database restore operations are not affected by the state of other members, backup operations might have to wait for a short duration while member crash recovery is completed on an offline and inconsistent member.

*Table 19. Effect of the state of other members in a Db2 pureScale instance on database backup and restore operations*

Operation	State of other members	
	Offline and consistent	Offline and inconsistent
Online backup	The backup operation succeeds. The other member cannot become active while the backup utility is accessing the log file header (LFH) near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.	The backup operation succeeds, but it must wait for member crash recovery to be completed and for the other member to become either active or consistent. The other member cannot become active while the backup utility is accessing the LFH near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.
Restore	The restore operation is completed normally.	The restore operation is completed normally.

## Image and archive naming

File names for backup images that you create on disk consist of a concatenation of several elements, separated by periods:

*DB\_alias.Type.Inst\_name.DBPARTnnn.Timestamp.Seq\_num*

**DB\_alias**

The database alias name that you specified when you invoked the backup utility.

**Type** The type of backup operation, where 0 represents a full database backup, 3 represents a table space backup, and 4 represents a backup image generated by the **LOAD** command with the **COPY NO** option.

**Inst\_name**

The name of the current instance, which is the value of the **DB2INSTANCE** environment variable.

**nnn** The database partition number. In a Db2 pureScale environment, the number is always 000.

**Timestamp**

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

- *yyyy* represents the year.
- *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).

**Seq\_num**

A 3-digit number used as a file extension.

For example:

`SAMPLE.0.krodger.DBPART000.200802241234.001`

## Online backup with **INCLUDE LOGS**

An online backup operation with the **INCLUDE LOGS** option (the default) produces a backup image that includes the range of log files required to restore and roll the database forward to its minimum recovery time. If this backup image is then used to restore to a new database (perhaps during disaster recovery), and only the logs from the backup image are available during a subsequent rollforward operation, a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** parameter often returns an error message about a missing log file (SQL1273N). This is expected in some situations, because the database manager might have detected that additional logs were written after the backup operation, but that those logs are not available for the current rollforward operation. It might also be the case that one or more of the logs that are necessary to roll the database forward to a consistent point in time are missing. In either case, verify that the end point of the rollforward operation is acceptable and then issue a **ROLLFORWARD DATABASE** with the **AND STOP** parameter. If the rollforward operation has reached its minimum recovery time despite the missing log file, the **ROLLFORWARD DATABASE** with the **AND STOP** parameter should complete successfully; otherwise, it returns SQL1276N (the rollforward operation did not reach its minimum recovery time using this backup image).

## Disaster recovery and high availability through log shipping in a Db2 pureScale environment

*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. You can choose to keep a standby database up-to-date by applying the logs to it as they are archived, or you can keep the database or table space backup images and log archives on the standby site, and perform restore and rollforward operations only after a disaster has occurred. In either case, the rollforward operation on the standby site might detect that one or more log files are missing and return SQL1273N. Verify that the rollforward operation reached an acceptable time stamp, or take appropriate action to correct the problem.

If, during a log stream merge operation, the Db2 database manager determines that there is a missing log file in one of the log streams, an error is returned. The rollforward utility returns SQL1273N; the db2ReadLog API returns SQL2657N. If you choose to keep a standby database up-to-date by applying logs to it as they are archived, rollforward operations might frequently detect that some logs are missing.

Figure 18 shows an example of how two members could write log records to the log files in their active log stream. Each log file is represented by a box. Consider a scenario where both a primary and standby site have been set up for high availability. A **ROLLFORWARD DATABASE** command with the **END OF LOGS** option is attempted on the standby site at time points A, B and C. For any particular point in time, any log files that have been closed before that time have been archived and are accessible on the standby. Otherwise, the log file is still active on the primary and is not available to the standby yet (as shown for log file 4 on log stream 1 at time B).

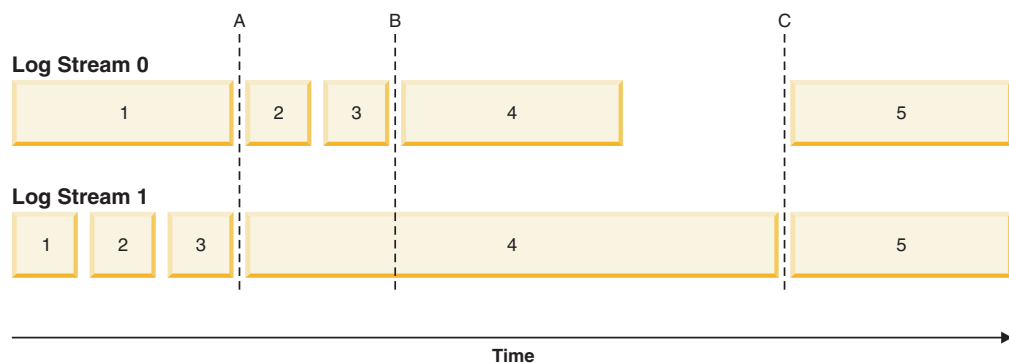


Figure 18. Log files in a Db2 pureScale environment

At time A, the **ROLLFORWARD DATABASE** command will complete successfully as log file 1 from log stream 0 was closed and archived at the same time as log file 3 from log stream 1. At time B however, the **ROLLFORWARD DATABASE** command will return v. This happens because at the time that the command is issued on the standby site, the standby site has access to log files 2 and 3 from log stream 0, but not to log file 4 from log stream 1 because the log file is still open and active on the primary site. Furthermore, since the log records in files 2 and 3 on log stream 0 were written during the same time period as the beginning of log file 4 on log stream 1, the rollforward operation cannot process log files 2 and 3 until log file 4 from log stream 1 is made available. At time C, when log file 4 is finally closed and archived on log stream 1, a **ROLLFORWARD DATABASE** command will complete successfully. It is possible to force the truncation and archiving of files across all



the log streams using the **ARCHIVE LOG** command, or by deactivating the database across all members. In the case of the **ARCHIVE LOG** command, the current log file on each log stream is truncated independently and there is no guarantee that it will happen at the exact same point in time across all members. Therefore, even if the **ARCHIVE LOG** command is issued, it is still possible to get an SQL1273N error when executing the **ROLLFORWARD DATABASE** command.

While missing log conditions are common and expected when using log shipping in a Db2 pureScale environment, in most cases, each rollforward operation on the standby will make additional progress over the last **ROLLFORWARD DATABASE** command (even when SQL1273N is returned) and therefore the error itself should often be expected. It is possible, however, for the primary site to have trouble archiving a file for one log stream while successfully archiving logs for the other log streams. This could be the result of a temporary problem accessing the archive storage for one log stream. Such problems can cause the log merge and replay on the standby to be held up, increasing the number of transactions that could be lost in the event of a disaster. To ensure that your standby system is up-to-date, issue a **ROLLFORWARD DATABASE** command with the **QUERY STATUS** parameter after each rollforward operation that returns SQL1273N and verify that progress is being made over time. If a rollforward operation on the standby is not making progress over an extended period of time, determine why the log file reported as missing is not available on the standby system and correct the problem. The **ARCHIVE LOG** command can be used to truncate the log files that are currently being updated on each member, making them eligible for archiving and subsequent replay on the standby system.

In the event of a disaster (for example, fire, earthquake, vandalism, or other catastrophic events) your plan for recovery might be to execute a rollforward operation through all remaining logs, or a restore and rollforward operation through all available logs. As mentioned previously, the rollforward operation might detect that one or more log file is missing, because log files were written on the primary but not yet archived at the time of the disaster (SQL1273N). It is also possible that a log that was archived cannot be found by the rollforward utility for some unexpected reason; this can also cause the rollforward utility to return SQL1273N. It is important to validate the end point of a rollforward operation by using the **ROLLFORWARD DATABASE** command with the **QUERY STATUS** parameter, and to decide whether or not the missing log condition is expected. If the missing log condition is expected, or the end point is acceptable, you can issue a **ROLLFORWARD DATABASE** command with the **STOP** parameter to complete the rollforward recovery process.

## Restrictions

After you drop a member, you cannot perform rollforward recovery operations through the point where the operation occurred. If you drop a member, the database is placed in backup pending state. You must perform either an incremental, or a full database backup operation before a connection to the database can be made. To recover, restore this backup image and roll forward to the end of the logs. If you must restore a backup image from before the topology change, you can roll forward only to the point at which the topology change occurred. This step can be accomplished by issuing a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** parameter (which returns SQL1546N) followed by a **ROLLFORWARD DATABASE** command with the **STOP** parameter. This operation will not recover any transactions that changed the database after the topology change.

In a Db2 pureScale environment, the **ON ALL DBPARTITIONNUMS** parameter and the **ON DBPARTITION (0)** parameter of the **BACKUP DATABASE** command are valid. If you specify a database partition number other than 0, however, an error (SQL0270N) is returned because no other database partitions exist.

## Examples

- Back up a four-member database named SAMPLE from any member:  
BACKUP DB SAMPLE
- Restore a one-member database named SAMPLE:  
RESTORE DB SAMPLE
- Use the **RECOVER DATABASE** command to restore and roll forward a database named SAMPLE from any member:  
RECOVER DB SAMPLE TO END OF LOGS

If the database does not exist, use the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands instead of the **RECOVER DATABASE** command because an existing database with a complete database history is required for the successful completion of the **RECOVER DATABASE** command.

## Monitoring backup operations

You can use the **LIST UTILITIES** command to monitor the progress of backup operations on a database.

### Procedure

Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter:

```
list utilities show detail
```

### Results

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

### Example

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

```
ID = 3
Type = BACKUP
Database Name = SAMPLE
Partition Number = 0
Description = offline db
Start Time = 08/04/2011 12:16:23.248367
State = Executing
Invocation Type = User
Throttling:
  Priority = Unthrottled
Progress Monitoring:
  Estimated Percentage Complete = 31
    Total Work = 123147277 bytes
    Completed Work = 37857269 bytes
    Start Time = 08/04/2011 12:16:23.248377
```

## Optimizing backup performance

When you perform a backup operation, the Db2 database manager automatically chooses an optimal value for the number of buffers, the buffer size, and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available, and the database configuration.

Therefore, depending on the amount of storage available on your system, consider allocating more memory by increasing the **util\_heap\_sz** configuration parameter.

The objective is to minimize the time it takes to complete a backup operation. Unless you explicitly enter a value for the following **BACKUP DATABASE** command parameters, the Db2 database manager selects one for them:

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

If the number of buffers and the buffer size are not specified, resulting in the Db2 database manager setting the values, it should have minimal effect on large databases. However, for small databases, it can cause a large percentage increase in backup image size. Even if the last data buffer written to disk contains little data, the full buffer is written to the image anyway. In a small database, this means that a considerable percentage of the image size might be empty.

You can also choose to do any of the following to reduce the amount of time required to complete a backup operation:

- Specify table space backup.

You can back up (and subsequently recover) part of a database by using the **TABLESPACE** option on the **BACKUP DATABASE** command. This facilitates the management of table data, indexes, and long field or large object (LOB) data in separate table spaces.

- Increase the value of the **PARALLELISM** parameter on the **BACKUP DATABASE** command so that it reflects the number of table spaces being backed up.

The **PARALLELISM** parameter defines the number of processes or threads that are started to read data from the database and to compress data during a compressed backup operation. Each process or thread is assigned to a specific table space, so there is no benefit to specifying a value for the **PARALLELISM** parameter that is larger than the number of table spaces being backed up. When it finishes backing up this table space, it requests another. Note, however, that each process or thread requires both memory and CPU overhead.

- Increase the backup buffer size.

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

- Increase the number of buffers.

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

- Use multiple target devices.

## Backup and restore statistics

Each successful backup and restore operation generates a single record in the **db2diag.log** file, which provides information on the performance of that operation. The log record is informational and is dumped at diaglevel 3 (the default) and 4.

## Example

The log records for backup and restore statistics consist of a row for each backup and restore buffer manipulator (db2bm) EDU and a row for each backup and restore media controller (db2med) EDU:

```
2012-07-30-15.41.30.012922-240 E15775E1464          LEVEL: Info
PID      : 15882                TID : 46913126656320  KTID : 16001
PROC     : db2sysc
INSTANCE: krodger              NODE : 000          DB   : SAMPLE
APPHDL   : 0-18                APPID: *LOCAL.krodger.120730194119
AUTHID   : KRODGER             HOSTNAME: hotel74
EDUID    : 49                  EDUNAME: db2agent (SAMPLE)
FUNCTION: Db2 UDB, database utilities, sqluxLogDataStats, probe:377
MESSAGE  : Performance statistics
DATA #1  : String, 951 bytes
```

```
Number of buffers = 4
Buffer size       = 16781312 (4097 4kB pages)
```

BM#	Total	I/O	MsgQ	WaitQ	Buffers	kBytes
---	-----	-----	-----	-----	-----	-----
000	6.30	0.02	0.00	6.18	4	640
001	5.88	4.48	0.00	1.33	9	139536
---	-----	-----	-----	-----	-----	-----
TOT	12.18	4.51	0.00	7.51	13	140176

MC#	Total	I/O	MsgQ	WaitQ	Buffers	kBytes
---	-----	-----	-----	-----	-----	-----
000	6.36	0.34	5.94	0.00	9	114748
001	6.29	0.18	5.60	0.10	6	81944
---	-----	-----	-----	-----	-----	-----
TOT	12.66	0.53	11.55	0.10	15	196692

The meanings of the various columns are as follows:

**BM** The db2bm EDU ID

**Total**

Length of time that each EDU existed

**I/O**

Time that was spent performing read or write I/O, to or from stable storage

**MsgQ**

Time that was spent waiting for an I/O buffer

**WaitQ**

Time that was spent waiting for a state machine control message.

**Buffers**

Number of I/O buffers that were processed

**kBytes**

Quantity of data that was processed

**MC** the db2med EDU ID

## Example

For compressed backups, the log record contains two additional columns for performance information about the compression operation:

```
2012-07-30-15.41.47.228766-240 E38419E1913          LEVEL: Info
PID      : 15882                TID : 46913126656320  KTID : 16081
PROC     : db2sysc
```

INSTANCE: krodger                      NODE : 000                      DB : SAMPLE  
 APPHDL : 0-29                      APPID: \*LOCAL.krodger.120730194132  
 AUTHID : KRODGER                      HOSTNAME: hotel74  
 EDUID : 80                      EDUNAME: db2agent (SAMPLE)  
 FUNCTION: Db2 UDB, database utilities, sqluxLogDataStats, probe:377  
 MESSAGE : Performance statistics  
 DATA #1 : String, 1399 bytes

Number of buffers = 4  
 Buffer size = 16781312 (4097 4K pages)

BM#	Total	I/O	Compr	MsgQ	WaitQ	Buffers	kBytes	Compr kBytes
---	-----	-----	-----	-----	-----	-----	-----	-----
000	12.08	4.36	7.18	0.00	0.37	5	139536	144941
001	11.87	0.01	0.01	0.00	11.79	1	640	640
---	-----	-----	-----	-----	-----	-----	-----	-----
TOT	23.96	4.38	7.19	0.00	12.17	6	140176	145581

MC#	Total	I/O		MsgQ	WaitQ	Buffers	kBytes	
---	-----	-----	-----	-----	-----	-----	-----	-----
000	12.07	0.11		11.76	0.00	4	49168	
001	12.10	0.07		11.84	0.15	4	32808	
---	-----	-----	-----	-----	-----	-----	-----	-----
TOT	24.18	0.19		23.61	0.15	8	81976	

#### Compr

Time that was spent performing the compression operation

#### Compr Bytes

Quantity of uncompressed data that was compressed

## Privileges, authorities, and authorization required to use backup

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

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.

## Compatibility of online backup and other utilities

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

The following utilities are compatible with online backup:

- **EXPORT**
- **INSPECT**

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

- **CREATE INDEX**

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

In DMS mode, online index create and online backup can run concurrently in most cases. There is a possibility if you have a large number of tables in the same tablespace as the one in which you are creating the index, that the online index create will internally acquire an online backup lock that will conflict with any concurrent online backup.

- **REORG INDEX with the ONLINE option**

As with online index create, in SMS mode, online index reorganization do not run concurrently with online backup due to the ALTER TABLE lock. Online index reorganization acquires it in exclusive mode while online backup acquires it in share. In addition, an online index reorganization operation, quiesces the table before the switch phase and acquires a Z lock, which prevents an online backup. However, the ALTER TABLE lock should prevent an online backup from running concurrently before the Z table lock is acquired.

In DMS mode, online index reorganization and online backup can run concurrently.

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

- **IMPORT**

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

- **TRUNCATE TABLE**

The TRUNCATE statement is not compatible with online backup because it gets a Z lock on the table and prevents an online backup from running concurrently.

- **LOAD**

Load operations that specify the **ALLOW READ ACCESS** parameter are not compatible with online backups when the **LOAD** command is issued with the **COPY NO** parameter. In this mode the utilities both modify the table space state, causing one of the utilities to report an error. Load operations that specify the **ALLOW READ ACCESS** parameter do not lead to an error if they also specify the **COPY YES** option. although there might still be some compatibility issues. In SMS mode, the utilities can execute concurrently, but they hold incompatible table lock modes and consequently might be subject to table lock waits. In DMS mode, the utilities both hold incompatible "Internal-B" (OLB) lock modes and might be subject to waits on that lock. If the utilities execute on the same table space concurrently, the load utility might be forced to wait for the backup utility to complete processing of the table space before the load utility can proceed.

- **REORG TABLE with the ONLINE option**

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

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

You cannot start an online backup of an SMS table space when a table within the same table space is being reorganized online. Both operations require an exclusive lock.

- **DDLs that require a Z lock (such as ALTER TABLE, DROP TABLE, and DROP INDEX)**

Online DMS table space backup is compatible with DDLs that require a Z lock. Online SMS table space backup must wait for the Z lock to be released.

- Storage group DDLs

If you are modifying the database storage groups by issuing one of the following statements, you should take care to coordinate this operation with your online backup schedule:

- CREATE STOGROUP
- ALTER STOGROUP
- DROP STOGROUP
- RENAME STOGROUP
- ALTER DATABASE

If there is an online backup in progress, the storage group DDL waits behind that operation until it can obtain the appropriate lock, which can potentially take a long time. Similarly, an online backup waits behind any in-progress storage group DDL, until that DDL is committed or rolled back.

- **RUNSTATS** with the ALLOW WRITE or ALLOW READ option

The **RUNSTATS** command is compatible with online backup except when the system catalog table space is an SMS table space. If the system catalog resides in an SMS table space, then the **RUNSTATS** command and the online backup hold incompatible table locks on the table causing lock waits.

- ALTER TABLESPACE

Operations that enable or disable autoresize, or alter autoresize containers, are not permitted during an online backup of a table space.

- ALTER TABLESPACE with the REBALANCE option

When online backup and rebalancer are running concurrently, online backup pauses the rebalancer and does not wait for it to complete.

The following utilities are not compatible with online backup:

- **REORG TABLE**
- **RESTORE DATABASE**
- **ROLLFORWARD DATABASE**
- **LOAD** with the **ALLOW NO ACCESS** option
- **SET WRITE**
- **BACKUP DATABASE** with the **ONLINE** option

This applies to database-level online backups and table-space-level online backups (if they involve the same table space or table spaces).

## Backup examples

This topic contains some examples of different backup strategies.

### Backup to TSM

In the following example database SAMPLE is backed up to a TSM server using 2 concurrent TSM client sessions. The backup utility will compute the optimal number of buffers. The optimal size of the buffers, in 4 KB pages, is automatically calculated based on the amount of memory and the number of target devices that are available. The parallelism setting is also automatically calculated and is based on the number of processors available and the number of table spaces to be backed up.

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

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

## Incremental backup

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
```

## Backup to tape

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

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

---

## Recover overview

The recover utility performs the necessary restore and rollforward operations to recover a database to a specified time, based on information found in the recovery history file.

When you use this utility, you specify that the database be recovered to a point-in-time or to the end of the log files. The utility will then select the best suitable backup image and perform the recovery operations.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for recovering databases. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see *Administering databases with task assistants*.

The recover utility does not support the following **RESTORE DATABASE** command options:

- **TABLESPACE** *tablespace-name*. Table space restore operations are not supported.
- **INCREMENTAL**. Incremental restore operations are not supported.
- **OPEN** *num-sessions* **SESSIONS**. You cannot indicate the number of I/O sessions that are to be used with TSM or another vendor product.
- **BUFFER** *buffer-size*. You cannot set the size of the buffer used for the restore operation.
- **DLREPORT** *filename*. You cannot specify a file name for reporting files that become unlinked.
- **WITHOUT ROLLING FORWARD**. You cannot specify that the database is not to be placed in rollforward pending state after a successful restore operation.
- **PARALLELISM** *n*. You cannot indicate the degree of parallelism for the restore operation.
- **WITHOUT PROMPTING**. You cannot specify that a restore operation is to run unattended



In addition, the recover utility does not allow you to specify any of the **REBUILD** options. However, the recovery utility will automatically use the appropriate **REBUILD** option if it cannot locate any database backup images based on the information in the recovery history file.

For the **RECOVER DATABASE** command, you cannot use the **TABLESPACE** option or the **INCREMENTAL** option from the **RESTORE DATABASE** command.

For the **RECOVER DATABASE** command, restore option is automated. Same applies for the **REBUILD** option in the **RESTORE** command.

## Recovering data

The **RECOVER DATABASE** command recovers a database and all storage groups to a specified time, by using information found in the recovery history file.

### Before you begin

If you issue the **RECOVER DATABASE** command following an incomplete recover operation that ended during the rollforward phase, the recover utility attempts to continue the previous recover operation, without redoing the restore phase. If you want to force the recover utility to redo the restore phase, issue the **RECOVER DATABASE** command with the **RESTART** option to force the recover utility to ignore any prior recover operation that failed to complete. If you are using the application programming interface (API), specify the caller action **DB2RECOVER\_RESTART** for the **iRecoverAction** field to force the recover utility to redo the restore phase.

If the **RECOVER DATABASE** command is interrupted during the restore phase, it cannot be continued. You must reissue the **RECOVER DATABASE** command.

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

### About this task

The database can be local or remote.

**Note:** In a partitioned database environment, the recover utility must be invoked from the catalog partition of the database.

### Procedure

To invoke the recover utility, use the:

- **RECOVER DATABASE** command, or
- db2Recover application programming interface (API).

### Example

The following example shows how to use the **RECOVER DATABASE** command through the CLP:

```
db2 recover db sample
```

## Recovering data using db2adutl

You can perform cross-node recovery using the **db2adutl** command, **logarchopt1** and **vendoropt** database configuration parameters. This recovery is demonstrated in examples from a few different Tivoli Storage Manager (TSM) environments.

For the following examples, computer 1 is called bar and is running the AIX operating system. The user on this machine is roecken. The database on bar is called zample. Computer 2 is called dps. This computer is also running the AIX operating system, and the user is regress9.

### Example 1: TSM server manages passwords automatically (PASSWORDACCESS option set to GENERATE)

This cross-node recovery example shows how to set up two computers so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed using the **PASSWORDACCESS=GENERATE** option.

**Note:** After updating the database configuration, you might have to take an offline backup of the database.

1. To enable the database for log archiving for the bar computer to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
bar:/home/roecken> db2 update db cfg for zample using LOGARCHMETH1 tsm
```

The following information is returned:

```
DB200000I The UPDATE DATABASE CONFIGURATION command completed successfully.
```

2. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

3. Verify that there are no applications connected to the database using the following command:

```
db2 list applications
```

You should receive a message that says that no data was returned.

**Note:** In a partitioned database environment, you must perform this step on all database partitions.

4. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

**Note:** In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see “Backing up data” on page 303.

5. Connect to the zample database using the following command:

```
db2 connect to zample
```

6. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
bar:/home/roecken> db2 load from mr of del modified by noheader replace
into employee copy yes use tsm
```

where in this example, the table is called `employee`, and the data is being loaded from a delimited ASCII file called `mr`. The **COPY YES** option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

**Note:** You can specify the **COPY YES** option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to **USEREXIT**, **LOGRETAIN**, **DISK**, or **TSM**.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N The utility is beginning to load data from file "/home/roecken/mr".

SQL3500W The utility is beginning the "LOAD" phase at time "02/16/2009
15:12:13.392633".

SQL3519W Begin Load Consistency Point. Input record count = "0".

SQL3520W Load Consistency Point was successful.

SQL3110N The utility has completed processing. "1" rows were read from the
input file.

SQL3519W Begin Load Consistency Point. Input record count = "1".

SQL3520W Load Consistency Point was successful.

SQL3515W The utility has finished the "LOAD" phase at time "02/16/2009
15:12:13.445718".
```

```
Number of rows read      = 1
Number of rows skipped   = 0
Number of rows loaded    = 1
Number of rows rejected  = 0
Number of rows deleted   = 0
Number of rows committed = 1
```

7. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the `zample` database:

```
bar:/home/roecken/sql1lib/adsm> db2adutl query db zample
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.
  1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
  1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
Taken at: 2009-02-16-15.10.38
```

8. To enable cross-node recovery, you must give access to the objects associated with the bar computer to another computer and account. In this example, give access to the computer dps and the user regress9 using the following command:

```
bar:/home/roecken/sql1lib/adsm> db2adutl grant user regress9
on nodename dps for db zample
```

The following information is returned:

Successfully added permissions for regress9 to access ZAMPLE on node dps.

**Note:** You can confirm the results of the **db2adutl** grant operation by issuing the following command to retrieve the current access list for the current node:

```
bar:/home/roecken/sql1lib/adsm> db2adutl queryaccess
```

The following information is returned:

Node	Username	Database Name	Type
DPS	regress9	ZAMPLE	A

Access Types: B - backup images L - logs A - both

9. In this example, computer 2, dps, is not yet set up for cross-node recovery of the zample database. Verify that there is no data associated with this user and computer on the TSM server using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adutl query db zample
```

The following information is returned:

```
--- Database directory is empty ---
Warning: There are no file spaces created by Db2 on the ADSM server
Warning: No Db2 backup images found in ADSM for any alias.
```

10. Query the TSM server for a list of objects for the zample database associated with user roecken and computer bar using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adutl query db zample nodename
bar owner roecken
```

The following information is returned:

```
--- Database directory is empty ---

Query for database ZAMPLE

Retrieving FULL DATABASE BACKUP information.
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
Taken at: 2009-02-16-15.10.38
```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the dps computer.

11. Restore the zample database from the TSM server to the dps computer using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options  
"-fromnode=bar -fromowner=roecken" without prompting
```

The following information is returned:

```
DB20000I The RESTORE DATABASE command completed successfully.
```

**Note:** If the zample database already existed on dps, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

12. Perform a roll-forward operation to apply the transactions recorded in the zample database log file when a new table was created and new data loaded. In this example, the following attempt for the roll-forward operation will fail because the roll-forward utility cannot find the log files because the user and computer information is not specified:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The command returns the following error:

```
SQL4970N Roll-forward recovery on database "ZAMPLE" cannot reach the  
specified stop point (end-of-log or point-in-time) because of missing log  
file(s) on node(s) "0".
```

Force the roll-forward utility to look for log files associated with another computer using the proper **logarchopt** value. In this example, use the following command to set the **logarchopt1** database configuration parameter and search for log files associated with user roecken and computer bar:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1  
"-fromnode=bar -fromowner=roecken"
```

13. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT  
"-fromnode=bar -fromowner=roecken"
```

14. You can finish the cross-node data recovery by applying the transactions recorded in the zample database log file using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
Rollforward Status
```

Input database alias	= zample
Number of members have returned status =	1

Member number	Rollforward status	Next log to be read	Log files processed	Last committed transaction
-----	-----	-----	-----	-----
0	not pending		S0000000.LOG-S0000000.LOG	2009-05-06-15.28.11.000000 UTC

```
DB20000I The ROLLFORWARD command completed successfully.
```

The database zample on computer dps under user regress9 has been recovered to the same point as the database on computerbar under user roecken.

## Example 2: Passwords are user-managed (PASSWORDACCESS option set to PROMPT)

This cross-node recovery example shows how to set up two computers so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed by the users. In these environments, extra information is required, specifically the TSM nodename and password of the computer where the objects were created.

1. Update the client `dsm.sys` file by adding the following line because computer bar is the name of the source computer

```
NODENAME bar
```

**Note:** On Windows operating systems, this file is called the `dsm.opt` file. When you update this file, you must reboot your system for the changes to take effect.

2. Query the TSM server for the list of objects associated with user roecken and computer bar using the following command:

```
dps:/home/regress9/sql/lib/adsm> db2adutl query db zample nodename bar  
owner roecken password *****
```

The following information is returned:

Query for database ZAMPLE

```
Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving LOAD COPY information.  
1 Time: 20090216151213
```

```
Retrieving LOG ARCHIVE information.  
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,  
Taken at: 2009-02-16-15.10.38
```

3. If the `zample` database does not exist on computer `dps`, perform the following steps:

- a. Create an empty `zample` database using the following command:

```
dps:/home/regress9> db2 create db zample
```

- b. Update the database configuration parameter **`tsm_nodename`** using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
```

- c. Update the database configuration parameter **`tsm_password`** using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using  
tsm_password *****
```

4. Attempt to restore the `zample` database using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options
"-fromnode=bar -fromowner=roecken'" without prompting
```

The restore operation completes successfully, but a warning is issued:

```
SQL2540W Restore is successful, however a warning "2523" was
encountered during Database Restore while processing in No
Interrupt mode.
```

5. Perform a roll-forward operation using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

In this example, because the restore operation replaced the database configuration file, the roll-forward utility cannot find the correct log files and the following error message is returned:

```
SQL1268N Roll-forward recovery stopped due to error "-2112880618"
while retrieving log file "S00000000.LOG" for database "ZAMPLE" on node "0".
```

Reset the following TSM database configuration values to the correct values:

- a. Set the **tsm\_nodename** configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
```

- b. Set the **tsm\_password** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_password *****
```

- c. Set the **logarchopt1** database configuration parameter so that the roll-forward utility can find the correct log files using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
"-fromnode=bar -fromowner=roecken'"
```

- d. Set the **vendoropt** database configuration parameter so that the load recovery file can also be used during the roll-forward operation using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"-fromnode=bar -fromowner=roecken'"
```

6. You can finish the cross-node recovery by performing the roll-forward operation using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
Rollforward Status

Input database alias           = zample
Number of members have returned status = 1

Member number  Rollforward status  Next log to be read  Log files processed  Last committed transaction
-----
0              not pending      S00000000.LOG-S00000000.LOG  2009-05-06-15.28.11.000000 UTC

DB20000I The ROLLFORWARD command completed successfully.
```

The database **zample** on computer **dps** under user **regress9** has been recovered to the same point as the database on computer **bar** under user **roecken**

### Example 3: TSM server is configured to use client proxy nodes

This cross-node recovery example shows how to set up two computers as proxy nodes so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed using the **PASSWORDACCESS=GENERATE** option.

**Note:** After updating the database configuration, you might have to take an offline backup of the database.

In this example, the computers bar and dps are registered under the proxy name of clusternode. The computers are already setup as proxy nodes.

1. Register the computers bar and dps on the TSM server as proxy nodes using the following commands:

```
REGISTER NODE clusternode mypassword
GRANT PROXYNODE TARGET=clusternode AGENT=bar,dps
```

2. To enable the database for log archiving to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
bar:/home/roecken> db2 update db cfg for zample using
LOGARCHMETH1 tsm logarchopt1 "'-asnodename=clusternode'"
```

The following information is returned:

```
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
```

3. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

4. Verify that there are no applications connected to the database using the following command:

```
db2 list applications
```

You should receive a message that says that no data was returned.

**Note:** In a partitioned database environment, you must perform this step on all database partitions.

5. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm options "'-asnodename=clusternode'"
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

Instead of specifying the **-asnodename** option on the **BACKUP DATABASE** command, you can update the **vendoropt** database configuration parameter instead.

**Note:** In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see “Backing up data” on page 303.

6. Connect to the zample database using the following command:

```
db2 connect to zample
```

7. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
bar:/home/roecken> db2 load from mr of del modified by noheader
replace into employee copy yes use tsmwhere
```

where in this example, the table is called employee, and the data is being loaded from a delimited ASCII file called mr. The **COPY YES** option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.



**Note:** You can specify the **COPY YES** option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to USEREXIT, LOGRETAIN, DISK, or TSM.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N The utility is beginning to load data from file "/home/roecken/mr".
```

```
SQL3500W The utility is beginning the "LOAD" phase at time "02/16/2009
15:12:13.392633".
```

```
SQL3519W Begin Load Consistency Point. Input record count = "0".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3110N The utility has completed processing. "1" rows were read from the
input file.
```

```
SQL3519W Begin Load Consistency Point. Input record count = "1".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3515W The utility has finished the "LOAD" phase at time "02/16/2009
15:12:13.445718".
```

```
Number of rows read      = 1
Number of rows skipped   = 0
Number of rows loaded    = 1
Number of rows rejected  = 0
Number of rows deleted   = 0
Number of rows committed = 1
```

8. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the zample database:

```
bar:/home/roecken/sql1lib/adsm> db2adutl query db zample
options "-asnodename=clusternode"
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.
  1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.
No DELTA DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving LOAD COPY information.
  1 Time: 20090216151213
```

```
Retrieving LOG ARCHIVE information.
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
Taken at: 2009-02-16-15.10.38
```

9. In this example, computer 2, dps, is not yet set up for cross-node recovery of the zample database. Verify that there is no data associated with this user and computer using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adutl query db zample
```

The following information is returned:

```
--- Database directory is empty ---  
Warning: There are no file spaces created by Db2 on the ADSM server  
Warning: No Db2 backup images found in ADSM for any alias.
```

10. Query the TSM server for a list of objects for the zample database associated with the proxy node clusternode using the following command:

```
dps:/home/regress9/sql/lib/adsm> db2adutl query db zample  
options="-asnodename=clusternode"
```

The following information is returned:

```
--- Database directory is empty ---  
  
Query for database ZAMPLE  
  
Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1  
  
Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE  
  
Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE  
  
Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE  
  
Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE  
  
Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE  
  
Retrieving LOAD COPY information.  
1 Time: 20090216151213  
  
Retrieving LOG ARCHIVE information.  
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,  
Taken at: 2009-02-16-15.10.38
```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the dps computer.

11. Restore the zample database from the TSM server to the dps computer using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options  
"-asnodename=clusternode" without prompting
```

The following information is returned:

```
DB20000I The RESTORE DATABASE command completed successfully.
```

**Note:** If the zample database already existed on dps, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

12. Perform a roll-forward operation to apply the transactions recorded in the zample database log file when a new table was created and new data loaded. In this example, the following attempt for the roll-forward operation will fail because the roll-forward utility cannot find the log files because the user and computer information is not specified:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The command returns the following error:

```
SQL4970N Roll-forward recovery on database "ZAMPLE" cannot reach the  
specified stop point (end-of-log or point-in-time) because of missing log  
file(s) on node(s) "0".
```

Force the roll-forward utility to look for log files on another computer using the proper **logarchopt** value. In this example, use the following command to set the **logarchopt1** database configuration parameter and search for log files associated with user roecken and computer bar:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
"-asnodename=clusternode"
```

13. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"-asnodename=clusternode"
```

14. You can finish the cross-node data recovery by applying the transactions recorded in the zample database log file using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```

                                Rollforward Status

Input database alias              = zample
Number of members have returned status = 1

Member number  Rollforward status  Next log to be read  Log files processed  Last committed transaction
-----
              0      not pending          S0000000.LOG-S0000000.LOG  2009-05-06-15.28.11.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

The database zample on computer dps under user regress9 has been recovered to the same point as the database on computer bar under user roecken.

#### Example 4: TSM server is configured to use client proxy nodes in a Db2 pureScale environment

This example shows how to set up two members as proxy nodes so that you can recover data from one member to the other when log archives and backups are stored on a TSM server and where passwords are managed using the **PASSWORDACCESS=GENERATE** option.

**Note:** After updating the database configuration, you might have to take an offline backup of the database.

In this example, the members member1 and member2 are registered under the proxy name of clusternode. In Db2 pureScale environments, you can perform backup or data recovery operations from any member. In this example, data will be recovered from member2

1. Register the members member1 and member2 on the TSM server as proxy nodes using the following commands:

```
REGISTER NODE clusternode mypassword
GRANT PROXYNODE TARGET=clusternode AGENT=member1,member2
```

2. To enable the database for log archiving to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
member1:/home/roecken> db2 update db cfg for zample using
LOGARCHMETH1 tsm logarchopt1 "-asnodename=clusternode"
```

**Note:** In Db2 pureScale environments, you can set the global **logarchmeth1** database configuration parameters once from any member.

The following information is returned:

```
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
```

3. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

4. Verify that there are no applications connected to the database using the following command:

```
db2 list applications global
```

You should receive a message that says that no data was returned.

5. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm options '-asnodename=clusternode'
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

Instead of specifying the **-asnodename** option on the **BACKUP DATABASE** command, you can update the **vendoropt** database configuration parameter instead.

**Note:** In Db2 pureScale environments, you can run this command from any member to back up all data for the database.

6. Connect to the zample database using the following command:

```
db2 connect to zample
```

7. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
member1:/home/roecken> db2 load from mr of del modified by noheader replace  
into employee copy yes use tsmwhere
```

where in this example, the table is called `employee`, and the data is being loaded from a delimited ASCII file called `mr`. The **COPY YES** option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

**Note:** You can specify the **COPY YES** option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to **USEREXIT**, **LOGRETAIN**, **DISK**, or **TSM**.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N The utility is beginning to load data from file "/home/roecken/mr".
```

```
SQL3500W The utility is beginning the "LOAD" phase at time "02/16/2009  
15:12:13.392633".
```

```
SQL3519W Begin Load Consistency Point. Input record count = "0".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3110N The utility has completed processing. "1" rows were read from the  
input file.
```

```
SQL3519W Begin Load Consistency Point. Input record count = "1".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3515W The utility has finished the "LOAD" phase at time "02/16/2009  
15:12:13.445718".
```

```
Number of rows read          = 1
```

```

Number of rows skipped      = 0
Number of rows loaded       = 1
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1

```

8. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the zample database:

```

member1:/home/roecken/sql1ib/adsm> db2adutl query db zample
options "-asnodename=clusternode"

```

The following information is returned:

```

Retrieving FULL DATABASE BACKUP information.
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
Sessions: 1

```

```

Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

```

```

Retrieving DELTA DATABASE BACKUP information.
No DELTA DATABASE BACKUP images found for ZAMPLE

```

```

Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for ZAMPLE

```

```

Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

```

```

Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for ZAMPLE

```

```

Retrieving LOAD COPY information.
1 Time: 20090216151213

```

Retrieving LOG ARCHIVE information.

```

Log file: S0000000.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.01.10
Log file: S0000000.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.01.11
Log file: S0000000.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.01.19
Log file: S0000001.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.02.49
Log file: S0000001.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.02.49
Log file: S0000001.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.02.49
Log file: S0000002.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.03.15
Log file: S0000002.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.03.15
Log file: S0000002.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.03.16

```

9. Query the TSM server for a list of objects for the zample database associated with the proxy node clusternode using the following command:

```

member2:/home/regress9/sql1ib/adsm> db2adutl query db zample
options="-asnodename=clusternode"

```

The following information is returned:

```

--- Database directory is empty ---

```

Query for database ZAMPLE

```

Retrieving FULL DATABASE BACKUP information.
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
Sessions: 1

```

```

Retrieving INCREMENTAL DATABASE BACKUP information.
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

```

```

Retrieving DELTA DATABASE BACKUP information.

```

```

No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
1 Time: 20090216151213

Retrieving LOG ARCHIVE information.

Log file: S0000000.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.01.10
Log file: S0000000.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.01.11
Log file: S0000000.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.01.19
Log file: S0000001.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.02.49
Log file: S0000001.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.02.49
Log file: S0000001.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.02.49
Log file: S0000002.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.03.15
Log file: S0000002.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.03.15
Log file: S0000002.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.03.16

```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the member2 member.

10. Restore the zample database on the TSM server from the member2 member using the following command:

```

member2:/home/regress9> db2 restore db zample use tsm options
'-asnodename=clusternode' without prompting

```

The following information is returned:

```

DB20000I The RESTORE DATABASE command completed successfully.

```

**Note:** If the zample database already existed on member2, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

11. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```

member2:/home/regress9> db2 update db cfg for zample using VENDOROPT
"'-asnodename=clusternode'"

```

**Note:** In Db2 pureScale environments, you can set the global **vendoropt** database configuration parameters once from any member.

12. You can finish the cross-member data recovery by applying the transactions recorded in the zample database log file using the following command:

```

member2:/home/regress9> db2 rollforward db zample to end of logs and stop

```

The following information is returned:

```

Rollforward Status

Input database alias           = zample
Number of members have returned status = 3

```

Member number	Rollforward status	Next log to be read	Log files processed	Last committed transaction
0	not pending		S0000001.LOG-S0000012.LOG	2009-05-06-15.28.11.000000 UTC
1	not pending		S0000001.LOG-S0000012.LOG	2009-05-06-15.28.11.000000 UTC
2	not pending		S0000001.LOG-S0000012.LOG	2009-05-06-15.28.11.000000 UTC

DB20000I The ROLLFORWARD command completed successfully.

The database zample on member member2 under user regress9 has been recovered to the same point as the database on member member1 under user roecken.

## Recovering a dropped table

You might occasionally drop a table that contains data you still need. If so, 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. The restore and rollforward operations can be time-consuming if the database is large, and your data is unavailable during the recovery.

The dropped table recovery feature lets you recover your dropped table data by using table space-level restore and rollforward operations.

This table space-level recovery is faster than database-level recovery, and your database remains available to users.

## Before you begin

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 option can be enabled during table space creation, or by invoking the ALTER TABLESPACE statement. The DROPPED TABLE RECOVERY option is table space-specific and limited to regular table spaces. To determine if a table space is enabled for dropped table recovery, you can query the DROP\_RECOVERY column in the SYSCAT.TABLESPACES catalog table.

The dropped table recovery option is on by default when you create a table space. If you do not want to enable a table space for dropped table recovery, you can either explicitly set the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE statement, or you can use the ALTER TABLESPACE statement to disable dropped table recovery for an existing table space. If there are many drop table operations to recover, or if the history file is large, the dropped table recovery feature might have a performance impact on forward recovery.

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 re-create the table.

For partitioned tables, dropped table recovery is always on even if the dropped table recovery is turned off for non-partitioned tables in one or more table spaces. Only one dropped table log record is written for a partitioned table. This log record is sufficient to recover all the data partitions of the table.

## About this task

If the table was in reorg pending state when it was dropped, the CREATE TABLE DDL in the history file does not match exactly that of the import file. The import file is in the format of the table before the first **REORG**-recommended ALTER was performed, but the CREATE TABLE statement in the history file matches the state of the table including the results of any ALTER TABLE statements.

### File type modifiers to use with LOAD or IMPORT

To recover the table by loading or importing, specify the following file type modifiers:

- The file type modifier **usegraphiccodepage** should be used in the **IMPORT** or **LOAD** command if the data being recovered is of the GRAPHIC or VARGRAPHIC data type. The reason is that it might include more than one code page.
- The file type modifier **delprioritychar** should be used in the **IMPORT** or **LOAD** commands. It allows **LOAD** and **IMPORT** to parse rows which contains newline characters within character or graphic column data.

### Restrictions

Only one dropped table can be recovered at a time.

There are some restrictions on the type of data that is recoverable from a dropped table. It is not possible to recover:

- The DROPPED TABLE RECOVERY option cannot be used for temporary table.
- The metadata associated with row types. (The data is recovered, but not the metadata.) The data in the hierarchy table of the typed table is recovered. This data might contain more information than appeared in the typed table that was dropped.
- XML data. If you attempt to recover a dropped table that contains XML data, the corresponding column data is empty.

## Procedure

You can recover a dropped table by doing the following:

1. Identify the dropped table by invoking the **LIST HISTORY DROPPED TABLE** command. The dropped table ID is listed in the Backup ID column.
2. Restore a database- or table space-level backup image taken before the table was dropped.
3. Create an export directory to which files containing the table data are to be written. This directory must either be accessible to all database partitions, or exist on each database partition. Subdirectories under this export directory are created automatically by each database partition. These subdirectories are named **NODEnnnn**, 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, by using the **RECOVER DROPPED TABLE** parameter 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. Re-create the table by using the CREATE TABLE statement from the recovery history file.



6. Import the table data that was exported during the rollforward operation into the table. If the table was in reorg pending state when the drop took place, the contents of the CREATE TABLE DDL might need to be changed to match the contents of the data file.

## 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, committed, and written to disk, 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 19).

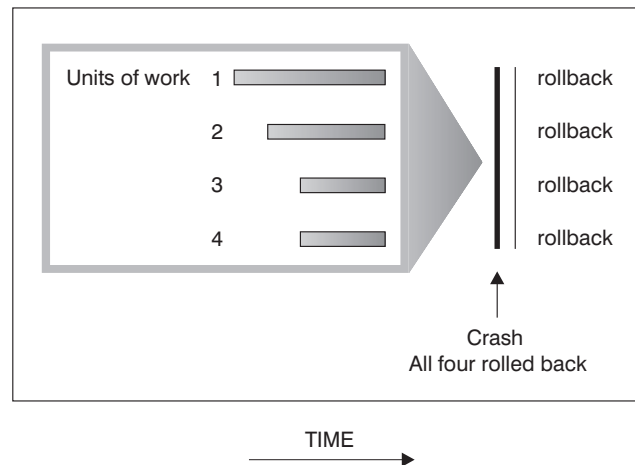


Figure 19. Rolling back units of work (crash recovery)

If the database or the database manager fails, the database can be left in an inconsistent state. The contents of the database might include changes made by transactions that were incomplete at the time of failure. The database might also be missing changes that were made by transactions that completed before the failure but which were not yet flushed to disk. A crash recovery operation must be performed in order to roll back the partially completed transactions and to write to disk the changes of completed transactions that were previously made only in memory.

Conditions that can necessitate a crash recovery include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down.
- A hardware failure such as memory, disk, CPU, or network failure.
- A serious operating system error that causes the Db2 instance to end abnormally.

If you want crash recovery to be performed automatically by the database manager, enable the automatic restart (**autorestart**) database configuration parameter by setting it to ON. (This is the default value.) If you do not want automatic restart behavior, set the **autorestart** database configuration parameter to OFF. As a result, you must issue the **RESTART DATABASE** command when a database

failure occurs. If the database I/O was suspended before the crash occurred, you must specify the **WRITE RESUME** option of the **RESTART DATABASE** command in order for the crash recovery to continue.

If you are using the IBM Db2 pureScale Feature, there are two specific types of crash recovery to be aware of: *member crash recovery* and *group crash recovery*. Member crash recovery is the process of recovering a portion of a database using the log stream of a single member after a member failure. Member crash recovery, which is usually initiated automatically as a part of a member restart, is an online operation—meaning that other members can still access the database. Multiple members can be undergoing member crash recovery at the same time. Group crash recovery is the process of recovering a database using multiple members' log streams after a failure that causes no viable cluster caching facility to remain in the cluster. Group crash recovery is also usually initiated automatically (as a part of a group restart) and the database is inaccessible while it is in progress, as with Db2 crash recovery operations outside of a Db2 pureScale environment.

If crash recovery occurs on a database that is enabled for rollforward recovery (that is, the **logarchmeth1** configuration parameter is not set to OFF), and an error occurs during crash recovery that is attributable to an individual table space, that table space is taken offline, and cannot be accessed until it is repaired. Crash recovery continues on other table spaces. At the completion of crash recovery, the other table spaces in the database are accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections are permitted. This behavior does not apply to Db2 pureScale environments. If an error occurs during member crash recovery or group crash recovery, the crash recovery operation fails.

If the database is configured for connectivity during crash recovery, the database might become connectable while crash recovery is in progress. Tables, indexes or objects that are still undergoing rollback will be locked in exclusive mode or super exclusive mode. For more information, see Database accessibility during backward phase of crash recovery or HADR takeover.

## 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.

### Note:

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

If the damaged table space is *not* the system catalog table space, Db2 attempts to make as much of the database available as possible.

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

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

## Recovering table spaces in recoverable databases

When crash recovery is necessary, the damaged table space is taken offline and is not accessible. It is placed in roll-forward pending state. If there are no additional problems, a restart operation succeeds in bringing the database online even with the damaged table space.

Once online, the damaged table space is unusable, but the rest of the database is usable. To fix the damaged table space and make it usable, use the following procedure.

### Procedure

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

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

**Note:** The rollforward operation first attempts to bring the table space from offline to normal state.

- Method 2
  1. Fix the damaged containers with or without losing the original data.
  2. Perform a table space restore operation.
  3. Complete a table space rollforward operation to the end of the logs or to a point-in-time.

## Recovering table spaces in non-recoverable databases

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

Therefore, the only valid action against such table spaces is to drop them.

### Procedure

To restart a database with damaged table spaces:

1. Invoke an unqualified restart database operation. The operation succeeds if there are no damaged table spaces. If it fails (SQL0290N), look in the administration notification log file for a complete list of table spaces that are currently damaged.

2. If you are willing to drop all of the damaged table spaces, initiate another restart database operation, listing all of the damaged table spaces under the DROP PENDING TABLESPACES option. If a damaged table space is included in the DROP PENDING TABLESPACES list, the table space is put into drop pending state, and you must drop the table space after the recovery operation is complete.

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

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

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

## Reducing the impact of media failure

*Media failure* is an error caused by defects on the read and write interfaces of storage devices, such as hard disk drives. This type of failure might be hard to detect and prevent, so you must take measures to reduce the impact of the failure if it occurs.

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

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

## Protecting against disk failure

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

A disk array is sometimes referred to simply as a RAID (Redundant Array of Independent Disks). Disk arrays can also be provided through software at the

operating system or application level. The point of distinction between hardware and software disk arrays is how CPU processing of input/output (I/O) requests is handled. For hardware disk arrays, I/O activity is managed by disk controllers; for software disk arrays, this is done by the operating system or an application.

## Hardware disk arrays

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

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

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

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

RAID level 1+0 (10) involves mirroring and striping the data across at least two disks. Mirroring writes the data to two or more disks at the same time which gives you the same fault tolerance as RAID level 1. Striping breaks the data into blocks and each block is written down to a separate disk drive. This achieves high I/O performance by spreading the I/O load across many channels and drives but RAID level 1+0 reduces the effective disk space in half as it mirrors all the data. RAID Level 10 requires a minimum of 4 drives to implement.

RAID level 0+1 is implemented as a mirrored array whose segments are RAID 0 arrays and has the same fault tolerance as RAID level 5. This gives high I/O rates by spreading the I/O load across many channels and drives. RAID level 0+1, however, is not to be confused with RAID level 1+0. A single drive failure will cause the whole array to become, in essence, a RAID level 0 array.

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

You might consider using a RAID level 1 hardware disk array or a software disk array for your logs, because this provides recoverability to the point of failure, and offers good write performance, which is important for logs. To do this, use the

*mirrorlogpath* configuration parameter to specify a mirror log path on a RAID level 1 file system. In cases where reliability is critical (because time cannot be lost recovering data following a disk failure), and write performance is not so critical, consider using a RAID level 5 hardware disk array. Alternatively, if write performance is critical, and the cost of additional disk space is not significant, consider a RAID level 1 hardware disk array for your data, as well as for your logs.

For detailed information about the available RAID levels, visit the following web site: [http://www.acnc.com/04\\_01\\_00.html](http://www.acnc.com/04_01_00.html)

## Software disk arrays


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

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

### CAUTION:

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

### Related information:

 Best practices: Database storage

## Reducing the impact of transaction failure

*Transaction failure* is the premature termination of transaction processing before transactions can be committed to the database, which might result in data loss or corruption.

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

- An uninterrupted power supply for each Db2 server
- Adequate disk space for database logs on all database partitions
- Reliable communication links among the database partition servers in a partitioned database environment
- Synchronization of the system clocks in a partitioned database environment.

## 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.

There are two types of database recovery:

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

**READ-ONLY**

No data change occurred at this server

**YES** Data change occurred at this server

**NO** Because of an error, the server is not prepared to commit

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

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

### **Transaction failure recovery on an active database partition server**

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

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.
- If the failed database partition server was the coordinator partition for the application, then agents that are still working for the application on the active servers are interrupted to do failure recovery. The transaction is rolled back locally on each database partition where the transaction is not in prepared state. On those database partitions where the transaction is in a prepared state, the transaction becomes an indoubt transaction. The coordinator database partition is not aware that the transaction is indoubt on some database partitions because the coordinator database partition is not available.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a

DISCONNECT message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator partition returns SQL0279.

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

### Transaction failure recovery on the failed database partition server

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

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

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

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

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

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

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

It is possible that crash recovery might not be able to resolve all the indoubt transactions. For example, some of the database partition servers might not be available. If the coordinator partition completes crash recovery before other database partitions involved in the transaction, crash recovery will not be able to resolve the indoubt transaction. This is expected because crash recovery is performed by each database partition independently. 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.

**Note:** In a partitioned database server environment, the RESTART database command is run on a per-node basis. In order to ensure that the database is restarted on all nodes, use the following recommended command:

```
db2_all "db2 restart database <database_name>"
```

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

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

- Db2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

## Identifying the failed database partition server

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

### SQL0279N

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

### SQL1224N

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

### SQL1229N

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

Determining which database partition server failed is a two-step process.

1. Find the partition server that detected the failure by examining the SQLCA. 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.)
2. Examine the administration notification log on the server found in step one for the node number of the failed server.

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

## Recovering from the failure of a database partition server

You can recover from a failed database partition server by identifying and correcting the issue that caused the failure.

## Procedure

To recover from the failure of a database partition server, perform the following steps.

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 mainframe or midrange servers

### Recovering indoubt transactions on the host when Db2 Connect has the Db2 Syncpoint Manager configured:

If your application has accessed a host or System i database server during a transaction, there are some differences in how indoubt transactions are recovered. To access host or System i database servers, Db2 Connect is used. The recovery steps differ if Db2 Connect has the Db2 Syncpoint Manager configured.

### About this task

The recovery of indoubt transactions at host or System i servers is normally performed automatically by the Transaction Manager (TM) and the Db2 Syncpoint Manager (SPM). An indoubt transaction at a host or System i server does not hold any resources at the local Db2 location, but does hold resources at the host or System i server as long as the transaction is indoubt at that location. If the administrator of the host or System i server determines that a heuristic decision must be made, then the administrator might contact the local Db2 database administrator (for example via telephone) to determine whether to commit or roll back the transaction at the host or System i 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.

## Procedure

The following steps can be used as a guideline for most situations involving an SNA communications environment:

1. Connect to the SPM. For example:

```
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 following example shows one indoubt transaction known to the SPM. The `db_name` is the local alias for the host or System i server. The `partner_lu` is the fully qualified luname of the host or System i server. This provides the best identification of the host or System i server, and should be provided by the caller from the host or System i server. The `luwid` provides a unique identifier for a transaction and is available at all hosts and System i servers. If the transaction in question is displayed, then the `uow_status` field can be used to

determine the outcome of the transaction if the value is C (commit) or R (rollback). If you issue the **LIST DRDA INDOUBT TRANSACTIONS** command with the **WITH PROMPTING** parameter, you can commit, roll back, or forget the transaction interactively.

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

3. If an indoubt transaction for the partner\_lu and for the luwid is not displayed, or if the **LIST DRDA INDOUBT TRANSACTIONS** command returns as follows, then the transaction was rolled back.

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

### What to do next

There is another unlikely but possible situation that can occur. If an indoubt transaction with the proper luwid for the partner\_lu is displayed, but the uow\_status is "I", the SPM does not 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 System i server based on the heuristic decision.

### Recovering indoubt transactions on the host when Db2 Connect does not use the Db2 Syncpoint Manager:

If your application has accessed a host or System i database server during a transaction, there are some differences in how indoubt transactions are recovered. To access host or System i database servers, Db2 Connect is used. The recovery steps differ if Db2 Connect has the Db2 Syncpoint Manager configured.

### About this task

Use the information in this section when TCP/IP connectivity is used to update Db2 for z/OS in a multisite update from the 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.

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.

## Procedure

1. On the z/OS host, issue the command `DISPLAY THREAD TYPE(INDOUBT)`.  
From this list identify the transaction that you want to heuristically complete.  
For details about the `DISPLAY` command, see the *Db2 for z/OS Command Reference*. The LUWID displayed can be matched to the same luwid at the Transaction Manager Database.
2. Issue the `RECOVER THREAD( <LUWID>) ACTION(ABORT|COMMIT)` command, depending on what you want to do.  
For details about the `RECOVER THREAD` command, see the *Db2 for z/OS Command Reference*.

## Database accessibility during backward phase of crash recovery or HADR takeover

If a database fails unexpectedly before all transactions (or units of work) are committed, or before the changes associated with committed transactions are fully written to disk, then the database is left in an inconsistent state. When a database restart is performed, these transactions are rectified during a process called crash recovery. During crash recovery, the changes associated with these transactions are first replayed from the transaction logs (we often refer to this as the Forward or Redo phase of crash recovery). Transactions which were not yet committed at the time of the database failure are then rolled back (we often refer to this as the Backward or Undo phase of crash recovery). A TAKEOVER HADR operation can also perform a Backward phase to rollback changes that were not yet committed at the time of the operation.

For releases prior to v11.1.2.2, the database is only accessible after the full completion of crash recovery or HADR Takeover. This is not ideal, since a very large transaction could require a lengthy Backward phase, leaving the database entirely inaccessible until it is completed.

Beginning in v11.1.2.2, the database can be configured to allow connectivity during the Backward phase of crash recovery. This behaviour can be enabled by using the following database registry variable:

```
db2set DB2_ONLINERECOVERY=YES
```

The registry variable is read at the beginning of crash recovery, it is not possible to enable or disable while crash recovery is actively running.

The point in time when the database becomes connectable during the Backward phase will depend on the characteristics of the workload. Uncommitted transactions with certain workload operations will be undone during the Backward phase without allowing database access :

- SQL Data Definition Language (DDL) statements.
- Operations which modify the database catalogs.
- Operations against column - organized tables.
- Workloads that generate some other log - records that are not yet supported .

We refer to this as the Synchronous portion of the Backward phase. Thus, the database will remain inaccessible during the Backward phase until the point in time of the very first of these unsupported operations.

**Note:** It may be possible that, if the oldest uncommitted transactions contain these unsupported operations, then the majority of the Backward phase will be performed synchronously without database access available

After completion of the Synchronous portion of the Backward phase, database access is allowed. We refer to this as the Asynchronous portion of the Backward phase.

For the duration of the Asynchronous portion of the Backward phase, while database access is allowed :

1. Any tables, indexes, or objects that are not associated with any uncommitted transactions are fully accessible during the asynchronous Backward phase.
2. Any tables, indexes, or objects that are associated with an uncommitted transaction are protected by an Exclusive (X) lock if DB2\_ONLINERECOVERY\_WITH\_UR\_ACCESS=YES, or a Super Exclusive (Z) lock if DB2\_ONLINERECOVERY\_WITH\_UR\_ACCESS=NO. As the Backward phase compensates the uncommitted transactions, this lock will be released when there are no more remaining uncommitted transactions referencing the table/object. Thus, prior to the release of these locks, these tables/objects are only accessible to queries using UR isolation level when DB2\_ONLINERECOVERY\_WITH\_UR\_ACCESS=YES, and completely inaccessible when DB2\_ONLINERECOVERY\_WITH\_UR\_ACCESS=NO.

If these tables also contain LONG or LOB fields, or are organized by MDC or ITC, they will always be protected by a Super Exclusive (Z) lock. These objects are never accessible during the Asynchronous Backward phase.

**Note:** The undo of these transactions may inherently require a Super Exclusive (Z) lock. For example, undoing utility operations such as LOAD), and that will always supersede the Exclusive (X) lock acquired during the Asynchronous portion when DB2\_ONLINERECOVERY\_WITH\_UR\_ACCESS=YES.

The **RESTART DATABASE** or **TAKEOVER HADR** command will return at the start of the Asynchronous Backward phase, when the database becomes connectable.

While the crash recovery operation is running, a **DEACTIVATE DB** operation will return an SQL1495W warning/error. A **db2stop** operation will return an SQL1025N error. Attempting to force the crash recovery application agent using **FORCE APPLICATION 0** will return an SQL0104N error. A **db2stop force** operation is allowed and will interrupt the crash recovery. A **TAKEOVER HADR** operation is also allowed during any phase of crash recovery.

If the INDEXREC configuration parameter is configured for index recreation at database restart time, then indexes will be recreated after the completion of the Asynchronous Backward phase.

**Note:** For pureScale instances, the backward phase of member crash recovery will continue to be performed synchronously without database accessibility on that member, because the database remains connectable on other members. Similarly for group crash recovery, the backward phase will continue to be performed synchronously without database accessibility.

### Monitoring Crash Recovery Progress:

The progress of crash recovery can be monitored during any phase using either of the "db2pd -recovery" option; the LIST UTILITIES SHOW DETAIL CLP command; the SNAPUTIL administrative view or SNAP\_GET\_UTIL table function. During the asynchronous backward phase, the MON\_GET\_UTILITY table function and the CHANGE HISTORY event monitor can also be used.

```

$ db2pd -db mydb1 -recovery
Recovery:
Recovery Status 7400010F40000000
Current Log S0000031.LOG
Current LSN 000000000006114D
Current LRI 0000000000000010000000000026E8000000000006114D
Current LSO 80969655
Job Type CRASH RECOVERY
Job ID 1
Job Start Time (1479058883) Sun Nov 13 12:41:23 2016
Job Description Crash Recovery
Invoker Type User
Total Phases 2
Current Phase 2
Progress:
Address PhaseNum Description StartTime CompletedWork TotalWork
0x078... 1 Forward Sun Nov 13 12:41:23 1290045 4 bytes 12900454 bytes
0x078... 2 Backward Sun Nov 13 12:49:27 0 bytes 12900454 bytes

```

The Forward phase of crash recovery (when transactions are replayed from the transaction logs) is displayed as the first phase, and its progress can be monitored using the Total Work and Completed Work values over time.

The Backward phase of crash recovery (where uncommitted transactions are rolled back or undone) is displayed after the first phase completes. Its progress can also be monitored by observing the Total Work and Completed Work values over time.

For example, if the CompletedWork value is observed 10mins apart, and the difference in the CompletedWorkvalue is 123,456 bytes during this duration of time, then we can easily extrapolate the time to complete the remaining work:

```

ObservationTime_inMins = 10mins
WorkDone_inBytes = 123,456bytes
RemainingWork_inBytes = TotalWork &ndash; CompletedWork;
TimeToCompletion_inMins = (RemaingWork_inBytes / WorkDone_inBytes) * ObservationTime_inMins.

```

While the Synchronous portion of the Backward phase is in progress, it is not possible to estimate when the Backward phase will enter the Asynchronous portion. When the Synchronous portion completes, an ADM1505I message is printed in the administration notification log ("ADM1505I Crash recovery has completed synchronous processing"), and the following message will be displayed in the db2diag.log:

```

2016-11-12-50.03.33.674725-300 I269681A541 LEVEL: Info
PID : 63373472 TID : 2058 KTID : 131596723
PROC : db2sysc
INSTANCE: db2inst1 NODE : 000 DB : MYDB1
APPHDL : 0-7 APPID: *LOCAL.dsciaraf.161114020311
AUTHID : DB2INST1 HOSTNAME: hotelaix2
EDUID : 2058 EDUNAME: db2agent (X)
FUNCTION: Db2 UDB, recovery manager, sqlpresr, probe:3170
DATA #1 : <preformatted>
Crash recovery synchronous phase completed. Next LSN is 000000000006113C.

```

If the Backward phase of crash recovery performs an Asynchronous portion, then the MON\_GET\_UTILITY table function will display an entry with a UTILITY\_TYPE value of 'ONLINERECOVERY' after the database opens up for connectivity .

For example:

```
$ db2 " SELECT COORD_MEMBER, APPLICATION_HANDLE AS APPHDL, SUBSTR(APPLICATION_NAME, 1, 30) AS APPN
SUBSTR(SESSION_AUTH_ID, 1, 10) AS USER, UTILITY_TYPE, UTILITY_INVOKER_TYPE, SUBSTR(UTILITY_DETAIL,
COORD_MEMBER APPHDL APPNAME USER UTILITY_TYPE UTILITY_INVOKER_TYPE CMD
-----
0 21 db2undo_trans - ONLINERECOVERY AUTO RESTART DATABASE MYDB1
1 record(s) selected.
```

The CHANGE HISTORY event monitor is also capable of recording the start and end of the asynchronous portion of the Backward phase of crash recovery using an event-control type called ONLINERECOVERY.

## Monitoring applications during the Backward phase of crash recovery

After database access is allowed during the Backward phase of crash recovery, monitoring of applications should be done using traditional methods, with the following considerations:

All uncommitted transactions that are part of the Backward phase of crash recovery will have an Application Handle (AppHandle) of zero (0). These transactions will also have a transaction state of 'ABORT' and Tflag2 bit 0x00000001 set.

For example:

```
$ db2pd -db MYDB1 -transactions
Transactions:
Address AppHandle ... TranHdl ... State Tflag Tflag2 ... LogSpace ...
0x... 0 ... 3 ... ABORT 0x00000000 0x00000001 ... 780 ...
0x... 0 ... 4 ... ABORT 0x00000000 0x00000001 ... 9465875 ...
0x... 7 ... 5 ... READ 0x00000000 0x00000000 ... 0 ...
0x... 8 ... 6 ... READ 0x00000000 0x00000000 ... 0 ...
```

Any tables, indexes, or objects that were associated with an uncommitted transaction are protected by an Exclusive (X) or Super Exclusive (Z) lock for the duration of the Backward phase of crash recovery. This means that applications which attempt to access these objects during the Backward phase of crash recovery are subject to lock - wait conditions.

Lock - wait conditions can be monitored using the MON\_GET\_APPL\_LOCKWAIT table function , or the db2pd -wlocks option. For example:

```
$ db2 "select LOCK_OBJECT_TYPE as TYPE, LOCK_NAME, LOCK_MODE as MODE,
LOCK_MODE_REQUESTED as MODE_REQ, LOCK_STATUS as STATUS,
HLD_APPLICATION_HANDLE as HLD_APPHANDL, REQ_APPLICATION_HANDLE as REQ_APPHANDL
from TABLE (MON_GET_APPL_LOCKWAIT(NULL, -2))"
TYPE LOCK_NAME MODE MODE_REQ STATUS HLD_APPHANDL REQ_APPHANDL
-----
TABLE 0200050000000000000000000000000054 X IS W 0 7
1 record(s) selected.
```

In this example, Application Handle 7 is requesting an Intent Share (IS) mode lock on a Table, which is already held in Exclusive (X) mode by Application Handle 0 (an uncommitted transaction that is part of crash recovery).

Additional lock details can be observed using the MON\_FORMAT\_LOCK\_NAME table function. For example:

```
$ db2 "SELECT SUBSTR(TABNAME,1,20) AS TABNAME, SUBSTR(VALUE,1,50) AS
VALUE FROM TABLE(MON_FORMAT_LOCK_NAME('02000500000000000000000000000054')) as LOCK"
NAME VALUE
-----
```

```

LOCK_OBJECT_TYPE    TABLE
TBSP_NAME           TSPA
TABSCHEMA           MYSCHEM
A1TABNAME           MYTABLE3
4 record(s) selected.

```

In this example, the lock wait condition is associated with a table lock on table MYSCHEMA1.MYTABLE3 in tablespace TSPA.

Additional details about the application in lock - wait state can be obtained using the WLM\_GET\_WORKLOAD\_OCCURRENCE\_ACTIVITIES table function:

```

db2 "SELECT APPLICATION_HANDLE as APPHNDL,
LOCK_WAIT_TIME, SUBSTR(STMT_TEXT,1,45) AS STMT_TEXT
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(NULL,-1)) AS T"
APPHNDL    LOCAL_START_TIME    LOCK_WAIT_TIME    STMT_TEXT
-----

```

```

10    2016-11-13-12.55.35.821373    0    SELECT APPHNDL,LOCK_WAIT_TIME
7     2016-11-13-12.53.33.777571    27    select count(*) from t2
2 record(s) selected.

```

## 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
- Offsite storage of either database backups, table space backups, or both, as well as archived logs.

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

You can choose to keep a standby database up to date by applying the logs to it as they are archived. Or, you can choose to keep the database or table space backups and log archives in the standby site, and perform restore and rollforward operations only after a disaster has occurred. (In the latter case, recent backup images are preferable.) In a disaster situation, however, it is generally not possible to recover all of the transactions up to the time of the disaster.

The usefulness of a table space backup for disaster recovery depends on the scope of the failure. Typically, disaster recovery is less complicated and time-consuming if you restore the entire database; therefore, a full database backup should be kept at a standby site. If the disaster is a damaged disk, a table space backup of each table space on that disk can be used to recover. If you have lost access to a container because of a disk failure (or for any other reason), you can restore the container to a different location.

Another way you can protect your data from partial or complete site failures is to implement the Db2 high availability disaster recovery (HADR) feature. Once it is



set up, HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby.

You can also protect your data from partial or complete site failures using replication. Replication allows you to copy data on a regular basis to multiple remote databases. Db2 database provides a number of replication tools that allow you to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied.

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

Db2 database products provide you with several options when planning for disaster recovery. Based on your business needs, you might decide to use table space or full database backups as a safeguard against data loss, or you might decide that your environment is better suited to a solution like HADR. Whatever your choice, you should test your recovery procedures in a test environment before implementing them in your production environment.

## Version recovery

*Version recovery* is the restoration of a previous version of the database, using an image that was created during a backup operation.

You use this recovery method with non-recoverable databases (that is, databases for which you do not have archived logs). You can also use this method with recoverable databases by using the **WITHOUT ROLLING FORWARD** option on the **RESTORE DATABASE** command.

A database restore operation will restore the entire database using a backup image created earlier. A database backup allows you to restore a database to a state identical to the one at the time that the backup was made. However, every unit of work from the time of the backup to the time of the failure is lost (see Figure 20).

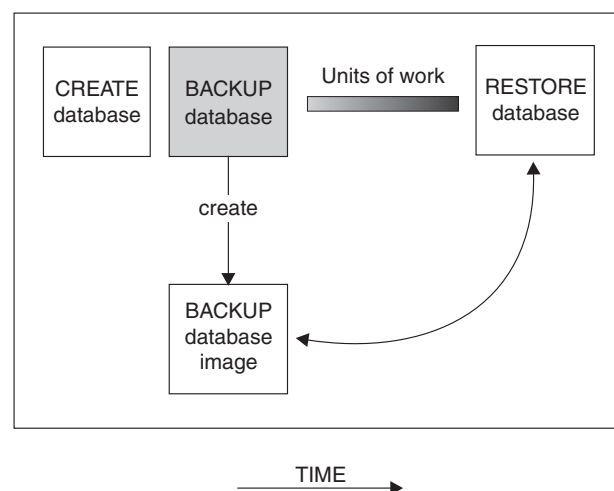


Figure 20. Version Recovery

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

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

## Rollforward recovery

You can complete a rollforward recovery for databases or table spaces.

To use the *rollforward recovery* method, you must have taken a backup of the database and archived the logs (by setting the **logarchmeth1** and **logarchmeth2** configuration parameters to a value other than OFF). Restoring the database and specifying the **WITHOUT ROLLING FORWARD** parameter 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** parameter for the restore database operation, the database will be in rollforward pending state at the end of the restore operation. This allows rollforward recovery to take place.

**Note:** The **WITHOUT ROLLING FORWARD** parameter cannot be used if:

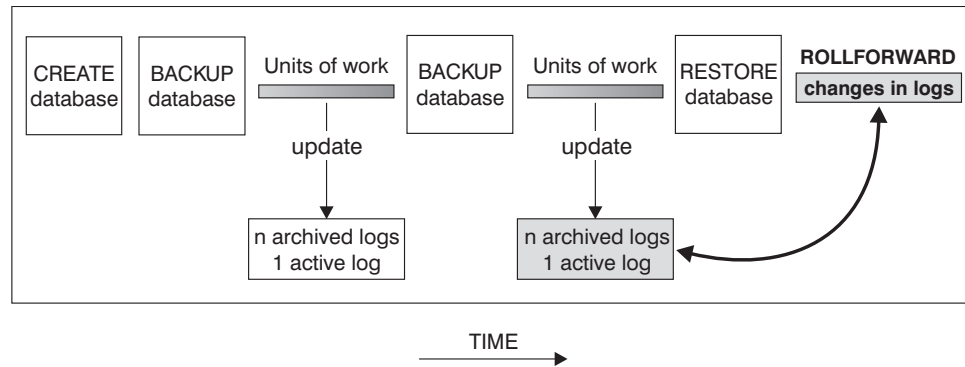
- You are restoring from an online backup image
- You are issuing a table space-level restore

During a recovery, archived log files are retrieved from the archive. If your archived log files are compressed, the files are automatically uncompressed and used. The archived log files are also automatically uncompressed when they are encountered in the active log path or overflow log path, if you manually copied the files there.

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 21 on page 359). The database logs record all changes made to the database. This method completes the recovery of the database to its state at a particular point in time, or to its state immediately before the failure (that is, to the end of the active logs).

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



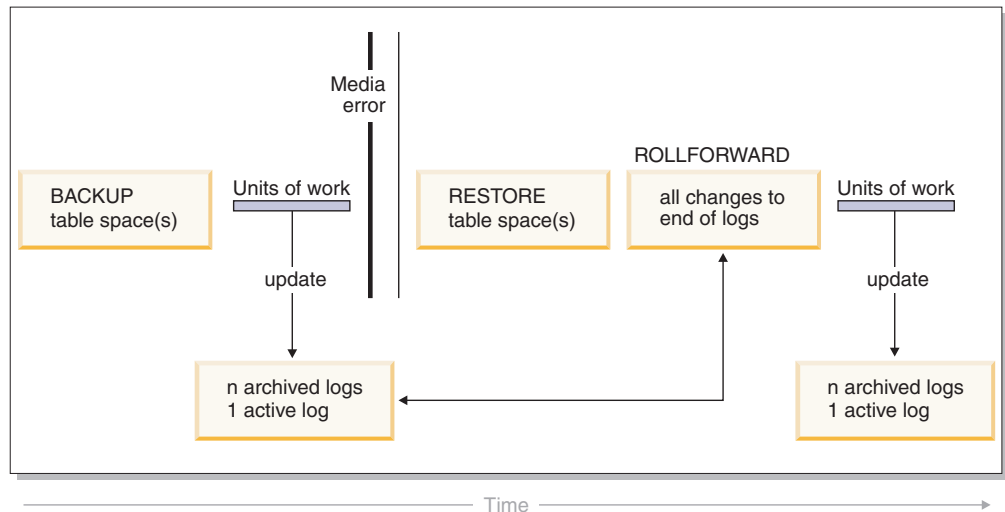
**Figure 21. Database Rollforward Recovery.** There can be more than one active log in the case of a long-running transaction.

- *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 22 on page 360). To perform a table space restore and rollforward operation, you need a backup image of either the entire database (that is, all of the table spaces), or one or more individual table spaces. You also need the log records that affect the table spaces that are to be recovered. You can roll forward through the logs to one of two points:
  - The end of the logs; or,
  - A particular point in time (called *point-in-time recovery*).

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

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

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



**Figure 22. Table Space Rollforward Recovery.** There can be more than one active log in the case of a long-running transaction.

In a partitioned database environment, if you are rolling a table space forward *to a point in time*, you do not have to supply the list of database partitions on which the table space resides. The Db2 database manager submits the rollforward request to all database partitions. This means the table space must be restored on all database partitions on which the table space resides.

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

Table space rollforward operations behave differently in a Db2 pureScale environment. For more information, see “Log stream merging and log file management in a Db2 pureScale environment” on page 278 and “Log sequence numbers in Db2 pureScale environments” on page 283.

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

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

## Storage group modifications during rollforward recovery

Whether storage group path modifications are redone during a rollforward operation depends on whether you redirected the storage group during the restore process. If you did not redefine a storage group during the database restore operation, log records affecting the storage group or its paths are replayed during

rollforward recovery. Storage path updates, storage group rename operations, and table space storage group association updates that are described in the log records are applied during the rollforward operation. If a rollforward operation is attempting to replay a log record related to adding storage paths or creating a storage group and a storage path cannot be found, error SQL1051N is returned.

If you redefined storage paths during the restore operation, the rollforward operation does not redo any changes to storage paths or media attributes of storage groups whose paths you redirected. However, changes to the data tag or name of storage groups are redone. Also, log records for other operations, including DROP STOGROUP operations, are replayed. It is assumed that any explicitly specified storage group paths have been set to their desired final paths.

If a rebalance operation is encountered in the log, table space rebalance operations are initiated during rollforward recovery. The rebalance operations might not be completed while the rollforward operation is in progress. In that case, the rebalance processing is suspended at the completion of the rollforward operation and is restarted the next time that you activate the database.

During a rollforward operation, if a CREATE STOGROUP statement is encountered in the log, the storage group is created on the paths that you specified when you issued the CREATE STOGROUP statement.

## Incremental backup and recovery

As the size of databases, and particularly warehouses, continues to expand into the terabyte and petabyte range, the time and hardware resources required to back up and recover these databases is also growing substantially.

Full database and table space backups are not always the best approach when dealing with large databases, because the storage requirements for multiple copies of such databases are enormous.

Consider the following issues:

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

To address these issues, Db2 provides incremental backup and recovery.

An *incremental backup* is a backup image that contains only pages that have been updated since the previous backup was taken. In addition to updated data and index pages, each incremental backup image also contains all of the initial database metadata (such as database configuration, table space definitions, database history, and so on) that is normally stored in full backup images.

### Note:

1. If a table space contains long field or large object data and an incremental backup is taken, all of the long field or large object data will be copied into the backup image if any of the pages in that table space have been modified since the previous backup.

2. If you take an incremental backup of a table space that contains a dirty page (that is, a page that contains data that has been changed but has not yet been written to disk) then all large object data is backed up. Normal data is backed up only if it has changed.
3. Data redistribution might create table spaces for all new database partitions if the **ADD DBPARTITIONNUMS** parameter on the **REDISTRIBUTE DATABASE PARTITION GROUP** command is specified; this can affect incremental backup operations.

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 noncumulative, backup image. The predecessor of a delta backup image is the most recent successful backup containing a copy of each of the table spaces in the delta backup image.

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

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

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

To enable the tracking of database updates, Db2 supports a new database configuration parameter, **trackmod**, which can have one of two accepted values:

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

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

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

## Restoring from incremental backup images

A restore operation from incremental backup images consists of four steps.

### About this task

1. Identifying the incremental target image.  
Determine the final image to be restored, and request an incremental restore operation from the Db2 restore utility. This image is known as the target image of the incremental restore, because it is the last image to be restored. The incremental target image is specified using the **TAKEN AT** parameter in the **RESTORE DATABASE** command.
2. Restoring the most recent full database or table space image to establish a baseline against which each of the subsequent incremental backup images can be applied.
3. Restoring each of the required full or table space incremental backup images, in the order in which they were produced, on top of the baseline image restored in Step 2.
4. Repeating Step 3 until the target image from Step 1 is read a second time. The target image is accessed twice during a complete incremental restore operation. During the first access, only initial data is read from the image; none of the user data is read. The complete image is read and processed only during the second access.

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

There are two ways to restore incremental backup images: automatic and manual:

- For an automatic incremental restore, the **RESTORE DATABASE** command is issued only once specifying the target image to be used. Db2 then uses the database history to determine the remaining required backup images and restores them.
- For a manual incremental restore, the **RESTORE DATABASE** command must be issued once for each backup image that needs to be restored (as outlined in the steps listed previously).

### Procedure

- To restore a set of incremental backup images using automatic incremental restore, issue the **RESTORE DATABASE** command specifying time stamp of the last image you want to restore with the **TAKEN AT** parameter, as follows:

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

This results in the restore utility performing each of the steps described at the beginning of this section automatically. During the initial phase of processing, the backup image with the specified time stamp (specified in the form *yyyymmddhhmmss*) is read, and the restore utility verifies that the database, its history, and the table space definitions exist and are valid.

During the second phase of processing, the database history is queried to build a chain of backup images required to perform the requested restore operation. If, for some reason this is not possible, and Db2 is unable to build a complete chain

of required images, the restore operation terminates, and an error message is returned. In this case, an automatic incremental restore is not possible, and you must issue the **RESTORE DATABASE** command with the **INCREMENTAL ABORT** parameter. This will clean up any remaining resources so that you can proceed with a manual incremental restore.

**Note:** It is highly recommended that you not use the **WITH FORCE OPTION** of the **PRUNE HISTORY** command. The default operation of this command prevents you from deleting history entries that might be required for recovery from the most recent, full database backup image, but with the **WITH FORCE OPTION**, it is possible to delete entries that are required for an automatic restore operation. During the third phase of processing, Db2 restores each of the remaining backup images in the generated chain. If an error occurs during this phase, you must issue the **RESTORE DATABASE** command with the **INCREMENTAL ABORT** option to clean up any remaining resources. You must then determine whether the error can be resolved before you reissue the **RESTORE DATABASE** command or attempt the manual incremental restore again.

- To restore a set of incremental backup images, using manual incremental restore, issue **RESTORE DATABASE** commands specifying time stamp of each image you want to restore with the **TAKEN AT** parameter, as follows:

1.

```
db2 restore database dbname incremental taken at timestamp
```

where *timestamp* points to the last incremental backup image (*the target image*) to be restored.

2.

```
db2 restore database dbname incremental taken at timestamp1
```

where *timestamp1* points to the initial full database (or table space) image.

3.

```
db2 restore database dbname incremental taken at timestampX
```

where *timestampX* points to each incremental backup image in creation sequence.

4.

Repeat Step 3, restoring each incremental backup image up to and including image *timestamp*.

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

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

## Limitations to automatic incremental restore

The automatic incremental restore is useful when you need to restore your database. However, you should consider the limitations of automatic incremental restore when you are deciding how you will recover your database to prevent unnecessary issues.

The following limitations affect automatic incremental restore:



1. If a table space name has been changed since the backup operation you want to restore from, and you use the new name when you issue a table space level restore operation, the required chain of backup images from the database history will not be generated correctly and an error will occur (SQL2571N).

Example:

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

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

Suggested workaround: Use manual incremental restore.

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

Example:

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

Suggested workarounds:

- Use manual incremental restore.
  - Restore the history file first from image <ts4> before issuing an automatic incremental restore.
3. If you restore a backup image from one database into another database and then do an incremental (delta) backup, you can no longer use automatic incremental restore to restore this backup image.

Example:

```
db2 create db a
db2 create db b

db2 update db cfg for a using trackmod on

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

db2 backup db b incremental -> ts2

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

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

Suggested workaround:

- Use manual incremental restore as follows:
- ```
db2 restore db b incremental taken at ts2
db2 restore db a incremental taken at ts1 into b
db2 restore db b incremental taken at ts2
```

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

## Optimizing recovery performance

There are strategies that you can use to improve Db2 performance during database recovery and decrease the time that is required to recover from a Db2 service outage.

The following should be considered when thinking about recovery performance:

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

You should also consider what other files are on the disk. For example, moving the logs to the disk used for system paging in a system that has insufficient real memory will defeat your tuning efforts.

Db2 database products automatically attempt to minimize the time it takes to complete a backup or restore operation by choosing an optimal value for the number of buffers, the buffer size and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available and the database configuration.

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

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

- The following apply for both backup and restore operations:
  - Multiple devices should be used.
  - Do not overload the I/O device controller bandwidth.
- Db2 database products use multiple agents to perform both crash recovery and database rollforward recovery. You can expect better performance during these operations, particularly on symmetric multi-processor (SMP) machines; using multiple agents during database recovery takes advantage of the extra CPUs that are available on SMP machines.

The agent type introduced by parallel recovery is **db2agnsc**. Db2 database managers choose the number of agents to be used for database recovery based on the number of CPUs on the machine.

Db2 database managers distribute 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.

- When you perform a recover operation, Db2 database managers will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration. Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the **util\_heap\_sz** configuration parameter.

## Privileges, authorities, and authorization required to use recover

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

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.

---

## Restore overview

You can restore the Db2 database to a previous state by using Db2 restore tools. A backup image of the database must exist before you can use these tools.

The simplest form of the Db2 **RESTORE DATABASE** command requires only that you specify the alias name of the database that you want to restore. For example:

```
db2 restore db sample
```

In this example, because the SAMPLE database exists and will be replaced when the **RESTORE DATABASE** command is issued, the following message is returned:

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

If you specify y, the restore operation should complete successfully.

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

A table space is not usable until the restore operation (possibly 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.

You can restore a database from a backup image taken on a 32-bit level into a 64-bit level, but not vice versa.

If you are restoring backups from 32-bit level environments to 64-bit level environments, review your database configuration parameters to ensure that they are optimized for the 64-bit instance environment. For example, the statement heap's default value is lower in 32-bit environments than in 64-bit environments.

The Db2 backup and restore utilities should be used to backup and restore your databases. Moving a fileset from one machine to another is not recommended as this may compromise the integrity of the database.

Under certain conditions, you can use transportable sets with the **RESTORE DATABASE** command to move databases. .

In IBM Data Studio Version 3.1 or later, you can use the task assistant for restoring database backups. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see *Administering databases with task assistants*.

## Using restore

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

### Before you begin

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 you want the new database to reside. Then, create the new database, specifying the code page and the territory of the server. The restore utility overwrites the code page of the destination database with the code page of the backup image.

### About this task

The database can be local or remote.

The following restrictions apply to the restore utility:

- During the **RESTORE**, if the database uses automatic storage, then the containers that are used on the automatic storage paths might be rebalanced. The rebalancing can happen when restoring a new or previously dropped database.

Rebalancing can also happen if there are configuration changes to the free space or the file system. For rebalancing conditions, see .

- You can only use the restore utility if the database has been previously backed up using the Db2 backup utility.
- If users other than the instance owner (on UNIX), or members of the DB2ADMNS or Administrators group (on Windows) attempt to restore a backup image, they get an error. If other users need access to the backup image, you need to change the file permissions the backup is generated.
- You cannot start a database restore operation while the rollforward process is running.
- If you do not specify the **TRANSPORT** option, then you can restore a table space into an existing database only if the table space currently exists, and if it is the same table space. In this situation, “same” means that the table space was not dropped and then re-created between the backup and the restore operation. The database on disk and in the backup image must be the same.
- You cannot issue a table space-level restore of a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.
- You cannot restore a backup taken in a single database partition environment into an existing partitioned database environment. Instead you must restore the backup to a single database partition environment and then add database partitions as required.
- When you are restoring a backup image with one code page into a system with a different codepage, the system code page is overwritten by the code page of the backup image.
- You cannot use the **RESTORE DATABASE** command to convert nonautomatic storage enabled table spaces to automatic storage enabled table space.
- You must verify that the server time was not reset or changed, and that the times associated with the members in multi-partitioned database environments are in sync. If the server time is not verified, the upgrade might return a SQL0440N error message.
- In order to change the database directory during restore, the database can not already exist.
- The following restrictions apply when you specify the **TRANSPORT** option:
  - If the backup image can be restored by a restore operation, and is supported for upgrades, then it can be transported.
  - If you are using an online backup, then both source and target data servers must be running the same Db2 version.
  - Issue the **RESTORE DATABASE** command against the target database. If the remote client is of the same platform as the server, then you can execute the schema transport locally on the server or through remote instance attachment. If a target database is a remote database cataloged in the instance where transport runs locally, then schema transport against that remote target database is not supported.
  - You can only transport tables spaces and schemas into an existing database. The transport operation does not create a new database. To restore a database into a new database, you can use the **RESTORE DATABASE** command without specifying the **TRANSPORT** option.
  - If the schemas in the source database are protected by any Db2 security settings or authorizations, then the transported schemas in the target database retain these same settings or authorizations.

- The **TRANSPORT** option is not supported in the Db2 pureScale environment, or in partitioned database environments.
- If any of the tables within the schema contains an XML column, the transport fails.
- The **TRANSPORT** option is incompatible with the **REBUILD** option.
- The **TRANSPORT** option is not supported for restore from a snapshot backup image.
- The staging database is created for transport. You cannot use it for other operations.
- The database configuration parameters on the staging table and the target table need to be the same, or the transport operation fails with an incompatibility error.
- The **auto\_reval** configuration parameter must be set to `deferred_force` on the target database to transport objects listed as invalid. Otherwise, the transport fails.
- If you use an online backup image and don't include the active logs, then the transport operation fails.
- If you use an online backup is used, then the backup image must have been created with the **INCLUDE LOGS** option
- If the backup image is from a previous release, it must be a full offline database level backup image.
- If an error occurs on either the staging or target database, you must reissue the entire restore operation. All failures that occur are logged in the `db2diag.log` file on the target server and should be reviewed before you reissue the **RESTORE DATABASE** command.
- If the transport client fails, then the staging database might not be properly cleaned up. In this case, you need to drop the staging database. Before reissuing the **RESTORE DATABASE** command, drop all staging databases to prevent containers of staging database from blocking subsequent transport.
- Concurrent transport running against the same target database is not supported.
- Generating a redirected restore script is not supported with table space transport.
- You can restore a table space if the storage group was updated. The target storage group during the table space restore is the storage group the table space is currently associated with when **RESTORE DATABASE** is executed.
- You cannot perform a point-in-time recovery to an earlier storage group association.
- Restoring a database that contains encrypted data in a Db2 instance that does not support encryption (where the IBM Global Security Kit is not installed) is supported. The encrypted data is included in the restored database, but you cannot access the encrypted data. To access the encrypted data, upgrade or restore the database to an instance that supports encryption.

## Procedure

To invoke the restore utility:

- Issue the **RESTORE DATABASE** command.
- Call the `db2Restore` application programming interface (API).
- Open the task assistant in IBM Data Studio for the **RESTORE DATABASE** command.

## Example

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

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

## Restoring from a snapshot backup image

Restoring from a snapshot backup uses the fast copying technology of a storage device to perform the data copying portion of the restore.

## Before you begin

To perform snapshot backup and restore operations, you need one of two things:

- A Db2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.
- For storage devices that are not supported, implement a custom script that allows your storage device to perform snapshot operations.

You must perform a snapshot backup before you can restore from a snapshot backup. See: “Performing a snapshot backup” on page 305.

## Procedure

You can restore from a snapshot backup using the **RESTORE DATABASE** command with the **USE SNAPSHOT** parameter, or the db2Restore API with the `SQLU_SNAPSHOT_MEDIA` media type:

- 

**RESTORE DATABASE** command:

```
db2 restore db sample use snapshot
```

- 

db2Restore API:

```
int sampleRestoreFunction( char dbAlias[],
                           char restoredDbAlias[],
                           char user[],
                           char pswd[],
                           char workingPath[] )
{
    db2MediaListStruct mediaListStruct = { 0 };

    rmediaListStruct.locations = &workingPath;
    rmediaListStruct.numLocations = 1;
    rmediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;

    db2RestoreStruct restoreStruct = { 0 };

    restoreStruct.piSourceDBAlias = dbAlias;
    restoreStruct.piTargetDBAlias = restoredDbAlias;
    restoreStruct.piMediaList = &mediaListStruct;
    restoreStruct.piUsername = user;
    restoreStruct.piPassword = pswd;
    restoreStruct.iCallerAction = DB2RESTORE_STORDEF_NOINTERRUPT;

    struct sqlca sqlca = { 0 };
```

```

        db2Restore(db2Version900, &restoreStruct, &sqlca);

    return 0;
}

```

### Restoring from a snapshot backup image with a script:

Using a custom script allows you to restore snapshot backup images taken using storage devices that are not supported by Db2 ACS.

#### Before you begin

You must have one of the following authorities: SYSADM, SYSCTRL, or SYSMAINT.

#### About this task

A snapshot restore operation restores a snapshot backup. You must use a custom script for that restore operation if your storage device does not provide a vendor library.

During snapshot restore operations, the protocol files that were written during the snapshot backup are read. As well, a new protocol file is written for the restore operation to show its progress. If the restore operation is successful, the protocol file is deleted; if the operation fails, you can use the protocol file to help investigate the cause of the failure.

A restore operation restores the latest image that matches the specified time stamp. For example, if there are two images for the time stamp 20121120, one taken at 201211201000 and one taken at 201211202000, the last one is chosen.

#### Restrictions

#### Procedure

To perform a snapshot restore:

1. Create a script that implements the Db2 ACS API. The script must be executable. For information on custom scripts, see “Db2 Advanced Copy Services (ACS) user scripts” on page 457.
2. Initiate the restore operation using either the **RESTORE DATABASE** command, the ADMIN\_CMD procedure with RESTORE DATABASE option, or the db2Restore API.

##### RESTORE DATABASE command

```

RESTORE DATABASE dbname
USE SNAPSHOT SCRIPT path-to-script
OPTIONS 'path-to-repository'
TAKEN AT timestamp LOGTARGET INCLUDE

```

##### ADMIN\_CMD procedure

```

CALL SYSPROC.ADMIN_CMD
    (restore database dbname
    use snapshot script path-to-script
    options 'path-to-repository'
    taken at timestamp logtarget include)

```

##### db2Restore API

```

int sampleRestoreFunction( char dbAlias[],
                           char restoredDbAlias[],
                           char user[],
                           char pswd[],

```



```

                                char workingPath[] )
{
    db2MediaListStruct mediaListStruct = { 0 };

    rmediaListStruct.locations = &workingPath;
    rmediaListStruct.numLocations = 1;
    rmediaListStruct.locationType = SQLU_SNAPSHOT_SCRIPT_MEDIA;

    db2RestoreStruct restoreStruct = { 0 };

    restoreStruct.piSourceDBAlias = dbAlias;
    restoreStruct.piTargetDBAlias = restoredDbAlias;
    restoreStruct.piMediaList = &mediaListStruct;
    restoreStruct.piUsername = user;
    restoreStruct.piPassword = pswd;
    restoreStruct.iCallerAction = DB2RESTORE_STORDEF_NOINTERRUPT;

    struct sqlca sqlca = { 0 };

    db2Restore(db2Version1050, &restoreStruct, &sqlca);

    return 0;
}

```

## Restoring to an existing database

For a database-level restore, the backup image can differ from the existing database in its alias name, its database name, or its database *seed*. A database seed is a unique identifier for a database that does not change during the life of the database.

The database manager assigns the seed when you create the database. Db2 always uses the seed from the backup image. You can restore a table space into an existing database only if the table space exists and if the table spaces are the same, meaning that you did not drop the table space and then re-create it between the backup and the restore operations. The database on disk and in the backup image must be the same. You cannot modify the currently defined storage groups or explicitly create new storage groups when restoring a table space.

Before you perform a **RESTORE DATABASE** on an existing image of the database, you must reset the **connect\_proc** parameter to NULL. If the **connect\_proc** is not set to NULL, you might encounter ERROR SQL0440N when you attempt a connection or rollforward command. To avoid this error, you must update the **connect\_proc** parameter to NULL by using the db2 update db cfg for <DATABASE> using connect\_proc NULL command.

When restoring to an existing database, the restore utility performs the following actions:

- 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 that you are restoring.
- Retains the recovery history file unless it is damaged or has no entries. If the recovery history file is damaged or contains no entries, the database manager copies the file from the backup image. If you want to replace the recovery history file, you can issue the **RESTORE DATABASE** command with the **REPLACE HISTORY FILE** parameter.
- Retains the authentication type for the existing database.
- Retains the database directories for the existing database. The directories define where the database is located and how it is cataloged.

- Compares the database seeds. If the seeds are different, the utility performs the following actions:
  - Deletes the logs that are associated with the existing database.
  - Copies the database configuration file from the backup image.
  - Sets the **NEWLOGPATH** parameter for the **RESTORE DATABASE** command to the value of the **logpath** database configuration parameter if you specified the **NEWLOGPATH** parameter.

If the database seeds are the same, the utility performs the following actions:

- Deletes all log files if the image is for a non-recoverable database.
- Deletes empty log files if the image is for a recoverable database. Non-empty log files are not affected.
- Retains the current database configuration file.
- Sets the **NEWLOGPATH** parameter for the **RESTORE DATABASE** command to the value of the **logpath** database configuration parameter if you specified the **NEWLOGPATH** parameter. Otherwise, the utility copies the current log path to the database configuration file. Validates the log path. If the database cannot use the path, the utility changes the database configuration to use the default log path.

## Restoring to a new database

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

When restoring to a new database, the restore utility:

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

## Using incremental restore in a test and production environment

Once a production database is enabled for incremental backup and recovery, you can use an incremental or delta backup image to create or refresh a test database.

You can do this by using either manual or automatic incremental restore.

To restore the backup image from the production database to the test database, use the **INTO *target-database-alias*** option on the **RESTORE DATABASE** command. For example, in a production database with the following backup images:

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

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

an example of a manual incremental restore would be:

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

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

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

If the database TEST already exists, the restore operation overwrites any data that is already there. If the database TEST does not exist, the restore utility creates it and then populates it with the data from the backup images.

Since automatic incremental restore operations are dependent on the database history, the restore steps change slightly based on whether the test database exists. To perform an automatic incremental restore to the database TEST, its history must contain the backup image history for database PROD. The database history for the backup image replaces any database history that already exists for database TEST if either of the following are true:

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

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

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

The restore utility creates the TEST database and populates it.

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

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

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

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

```
connect to test
Database Connection Information

Database server          = server_id
SQL authorization ID     = id
Local database alias     = TEST

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

connect reset
```

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

In this case, the **RESTORE DATABASE** command acts in the same manner as when the database TEST did not exist.

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

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

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

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

## Restore and roll forward through a topology change

A topology change involves adding members or dropping members from a Db2 pureScale instance (explicitly or through restore), or restoring between a Db2 pureScale instance and a Db2 Enterprise Server Edition instance. Some topology change scenarios also allow restore followed by rollforward recovery.

### Adding members to the topology

A strict superset Db2 pureScale environment refers to an environment where all members in the backup image are defined in the target member topology; and the target member topology contains additional members. For example, the backup source member topology includes members 0 and 1, and the target member topology contains members 0, 1 and 2.

#### Example 1 (Restore from a Db2 pureScale environment to a superset Db2 pureScale environment)

To restore the backup source member topology instance (containing two members) to the target member topology instance (containing three members):

On the source instance, back up the database:

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

On the target instance (Db2 pureScale instance that is a superset of the source member topology), restore database:

```
db2 restore database sample from /dev3/backup
```

After the restore, the database is usable on any of the members in the target instance. For example, to activate the database on all members, the **activate database** command can be run from any of the members:

```
db2 activate database sample
```

### **Example 2 (Restore a manual file copy Db2 pureScale instance to superset Db2 pureScale instance)**

If you have a Db2 pureScale instance with two members (for example 0 and 1), and you perform a manual file copy backup (such as FlashCopy). Then, you apply the backup to a Db2 pureScale instance with a superset topology (for example three members 0, 1 and 2), follow the steps in topic “Using a split mirror as a backup image” on page 104 or topic “Using a split mirror to clone a database in a Db2 pureScale environment” on page 101.

### **Dropping members from the topology**

These examples include an offline database backup image for a consistent database that is restored to a Db2 pureScale instance. However, the Db2 pureScale instance has a member topology that is not a superset of the source member topology in the backup image. This case could refer to strictly shrinking the topology (target member topology is a subset of the source member topology), both growing and shrinking (the intersection between the source and target member topology is not empty.) In all cases, there must be a common member between the source and target member topology. In addition, to ensure recoverability, following the restore you must perform either an incremental or full offline database backup.

A subset Db2 pureScale environment refers to an environment where not all members in the backup image are defined in the target member topology. For example, the backup source member topology includes members 0, 1, 2, and the target member topology contains members 0 and 1.

### **Example 3 (Restore from a Db2 pureScale environment to a subset Db2 pureScale environment)**

To restore the backup source member topology instance (containing three members) to the target member topology instance (containing two members):

On the source instance, to ensure that the database is in a consistent state, stop the instance and perform an offline backup:

```
db2stop
db2 backup database sample to /dev3/backup
```

On the target instance (Db2 pureScale instance that is a superset of the source member topology), restore database:

```
db2 restore database sample from /dev3/backup without rolling forward
```

Back up the database on the target instance after the topology change

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

or

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

#### Example 4 (Restore from a Db2 pureScale environment to a Db2 pureScale environment that is not superset)

To restore the backup source member topology instance (containing three members) to the target member topology instance (containing two members):

This example restores a source member topology (containing members 0, 1) to the target member topology instance (containing members 1 and 2). In this case, the only common member between the two topologies is member 1.

On the source instance, to ensure that the database is in a consistent state, stop the instance and perform an offline backup:

```
db2stop
db2 backup database sample to /dev3/backup
```

On the target instance, restore the database from member 1 (the common member). This causes a topology breaking event.

```
db2 restore database sample from /dev3/backup without rolling forward
```

You must perform either an incremental or full offline database backup of the database from member 1 (the common member) on the target instance.

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

or

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

#### Restore from Db2 pureScale Feature to Db2 Enterprise Server Edition

Starting in V10.5, the mobility of backup images back and forth between ESE instances and a Db2 pureScale instance is supported. For steps, see “Restore from Db2 pureScale Feature to Db2 Enterprise Server Edition” on page 417 or “Restore from Db2 Enterprise Server Edition to Db2 pureScale instance” on page 418.

#### Restore an online backup to a target Db2 pureScale environment that is not a superset

An online backup results in a database that is in an inconsistent state. A database in an inconsistent backup requires a rollforward operation. However, a rollforward operation through a drop member event is not supported. Running the restore command results in an error message.

#### Removing the offline backup requirement after a topology change running db2dart command

For recoverable databases, after a topology change you must take either an incremental or a full offline database backup for the database to be usable again. To ensure the recoverability of the database from that point in time, this recoverability is enforced by placing the databases in backup pending state.

A high risk workaround to this requirement, instead of running the **BACKUP DATABASE** command, you can run the **db2dart** command. However, any updates that are performed on the database are not recoverable until a full database backup is taken. This command has high risk of jeopardizing recoverability of the database, so it must be used only under the advisement of Db2 Service.

To remove the offline backup requirement:

```
db2dart sample /CHST /WHAT DBBP OFF
```

where WHAT DBBP OFF specifies database backup pending state is to be turned off.

## Restore plus rollforward recovery

The **ROLLFORWARD DATABASE** command supports the recovery through add member events. After the addition of a new member to a target topology, you do not need to take a database backup. Instead, you can restore a backup that is taken before the add member events took place. Then, perform a rollforward recovery to either the end of the transaction log or to any point in time after the add member events occurred. Inconsistent backups are also supported. Inconsistent backups are those backups when restored require a rollforward operation to bring the database back to a consistent point. In addition, a rollforward operation supports encountering add member events in the transaction logs and handle them transparently.

### Example 5 (Performing restore plus rollforward recovery through a member addition event)

The members that do not exist in the backup image could be added during the rollforward operation as the add member log record is found (this is referred to as an *explicit add*). Or if the add member log record (AMLR) is not found, members could be added on the subsequent connect to those members (this is referred to as an *implicit add*).

**Explicit add:** a full database backup B0 for database sample is taken on a Db2 pureScale instance with a topology that includes only member 0. After some time of processing read/write transactions for this database on member 0, member 1 is added. Clients continue processing transactions both in member 0 and member 1. The backup image B0 is restored to a cluster with two members 0 and 1. The user initiates a rollforward operation through the transaction logs. The rollforward operation encounters an add member log record for member 1 before you reach the end of the logs.

On the source instance (member 0), back up to image B0 with just member 0:

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

Continue processing transactions on member 0, then add member 1:

```
db2iupdt -m -add
```

Continue processing transactions on members 0 and 1.

On the target instance (with member 0 and 1), restore backup image B0 and transaction logs:

```
db2 restore database sample from /dev3/backup
```

Roll forward to end of logs:

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

The **ROLLFORWARD DATABASE** command encounters an add member log record (AMLR) for member 1 and processes the addition of the member. The rollforward operation reaches end of log and completes. The database is now usable.

**Implicit add:** a full database backup B0 for database sample is taken on a Db2 pureScale instance with a topology that includes only member 0. Process read/write transactions for this database on member 0. Backup image B0 is restored to a cluster with two members that is a superset of the source member topology: which includes members 0 and 1. Initiate a rollforward operation through the transaction logs, which does not find any events for member 1. In this case, after the rollforward completes, to use member 1, issue a **CONNECT** command (or **ACTIVATE DATABASE** command). Member 1 is added implicitly by either of these commands.

On the source instance (with member 0), back up to image B0:

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

Continue processing transactions on member 0.

On the target instance (with member 0 and 1), restore backup image B0 and transaction logs (note that all transaction logs are transferred as well, and potentially archived logs must be accessible on the target instance):

```
db2 restore database sample from /dev3/backup
```

Roll forward to end of logs:

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

The database is now usable.

**Implicit add at a point in time:** a full database backup B0 for database sample is taken on a Db2 pureScale instance with a topology that includes only member 0. After some time of processing read/write transactions for this database on member 0, backup image B0 is restored to a cluster with two members that is a superset of the source member topology. This superset includes members 0 and 1. Initiate a rollforward operation through the transaction logs to a point in time before the add member event (instead of rolling forward to end of logs).

On the source instance (with member 0), back up to image B0 with just member 0:

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

Continue processing transactions on member 0, then add member 1:

```
db2iupdt -m -add
```

Continue processing transactions on members 0 and 1.

On the target instance (with member 0 and 1), restore backup image B0 and transaction logs:

```
db2 restore database sample from /dev3/backup
```

Roll forward to a point in time before the addition of member 1:

```
db2 rollforward database sample to 2013-04-03-14.21.56 and stop
```

The database is now usable. On the first use new log chains are created for each of the members.



## Performing a redirected restore operation

A database restore operation uses a database backup image to recreate a database.

Use a redirected restore operation in any of the following situations:

- If you want to restore a backup image to a target machine that is different from the source machine
- If you want to restore your table space containers into a different physical location
- If your restore operation failed because one or more containers are inaccessible
- If you want to redefine the paths of a defined storage group

### Restrictions:

You cannot use a redirected restore to move data from one operating system to another.

You cannot create or drop a storage group during the restore process.

You cannot modify storage group paths during a table space restore process even if you are restoring all table spaces that are associated with the storage group.

The process for performing a redirected restore by using an incremental backup image is similar to the process of performing a redirected restore by using a non-incremental backup image. Use one of the following approaches:

- Issue the **RESTORE DATABASE** command with the **REDIRECT** parameter, and specify the backup image to use for the incremental restore of the database.
- Generate a redirected restore script from a backup image, and then modify the script as required.

Using the **RESTORE DATABASE** command approach is a two-step database restore process with an intervening step for defining a table space container or storage group path. To perform a redirected restore:

1. Issue the **RESTORE DATABASE** command with the **REDIRECT** parameter.
2. Take one of the following steps:
  - Define table space containers by issuing the **SET TABLESPACE CONTAINERS** command.
  - Define storage group paths for the database to be restored by issuing the **SET STOGROUP PATHS** command.
3. Issue the **RESTORE DATABASE** command again, this time specifying the **CONTINUE** parameter.

After you issue the **RESTORE CONTINUE** command, the new path takes effect as the table space container path for all associated table spaces. If you issue a **LIST TABLESPACE CONTAINERS** command or a **GET SNAPSHOT FOR TABLESPACES** command after the **SET STOGROUP PATHS** command and before the **RESTORE CONTINUE** command, the output for the table space container paths does not reflect the new paths that you specified by using the **SET STOGROUP PATHS** command.

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

Db2 database products provide SQL statements for adding, changing, or removing table space containers non-automatic-storage DMS table spaces, and storage group

paths of automatic storage table spaces. A redirected restore is the only way to modify a non-automatic-storage SMS table space container configuration.

You can redefine table space containers or modify storage group paths by issuing the **RESTORE DATABASE** command with the **REDIRECT** parameter.

Table space container redirection provides considerable flexibility for managing table space containers. You can alter the storage group configuration of a database before restoring any data pages from the backup image, similar to the way that you can redirect table space container paths. If you renamed a storage group since you produced the backup image, the storage group name that is specified by the **SET STOGROUP PATHS** command refers to the storage group name from the backup image, not the more recent name.

## Performing a redirected restore operation in a partitioned database environment

In a partitioned database environment, during a redirected database restore, you can redirect the storage group paths to new storage group paths only from the catalog database partition. For all other database partitions you must have their storage group paths synchronized with those of the catalog partition.

Modifying any storage group paths on the catalog partition places all non-catalog partitions into a **RESTORE\_PENDING** state. If you redirect storage group paths, you must restore the catalog partition before any other database partition. After you restore the catalog database partition, you can restore the non-catalog database partitions in parallel, without any storage group path redirection. The non-catalog database partitions automatically acquire the new storage group paths that you specified for the catalog database partition. New storage group paths are also automatically acquired when the storage group paths are implicitly changed during a database restore when you are restoring a different database (one with a different name, instance, or seed).

If you modified the storage group paths since taking the last backup, you can still use that backup image (with different storage group paths) for a restore on any database partition. This restore is not considered a redirected restore. Restoring from that backup image temporarily causes the database partition to use the storage group paths that you defined at the time that you created the backup. Perform a rollforward recovery to reapply the storage group path modifications and resynchronize all of the database partitions.

## Examples

### Example 1

You can perform a table space container redirected restore on database **SAMPLE** by using the **SET TABLESPACE CONTAINERS** command to define table space containers:

```
db2 restore db sample redirect without prompting
SQL1277W A redirected restore operation is being performed.
During a table space restore, only table spaces being restored can
have their paths reconfigured. During a database restore, storage
group storage paths and DMS table space containers can be reconfigured.
```

```
DB20000I The RESTORE DATABASE command completed successfully.
```

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

DB20000I The SET TABLESPACE CONTAINERS command completed successfully.

```
db2 restore db sample continue
```

DB20000I The RESTORE DATABASE command completed successfully.

### Example 2

You can redefine the paths of the defined storage group by using the **SET STOGROUP PATHS** command:

```
RESTORE DB SAMPLE REDIRECT
```

```
SET STOGROUP PATHS FOR sg_hot ON '/ssd/fs1', '/ssd/fs2'
```

```
SET STOGROUP PATHS FOR sg_cold ON '/hdd/path1', '/hdd/path2'
```

```
RESTORE DB SAMPLE CONTINUE
```

### Example 3

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

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

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

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example, in a Windows environment:

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

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command for every table space whose container locations are being redefined.

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

```
db2 restore db mydb continue
```

This is the final step of the redirected restore operation.

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

#### Note:

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

```
db2 restore db mydb abort
```

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

### Example 4

Following is a typical manual incremental redirected restore scenario for a database whose alias is MYDB and has the following backup images:

```
backup db mydb
```

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

```
backup db mydb incremental
```

Backup successful. The timestamp for this backup image is : <ts2>

1. Issue a RESTORE DATABASE command with the INCREMENTAL and REDIRECT options.

```
db2 restore db mydb incremental taken at <ts2> replace existing redirect
```

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

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

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

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

```
db2 restore db mydb continue
```
4. The remaining incremental restore commands can now be issued as follows:

```
db2 restore db mydb incremental taken at <ts1>
db2 restore db mydb incremental taken at <ts2>
```

This is the final step of the redirected restore operation.

**Note:**

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

```
db2 restore db mydb abort
```
2. After successful completion of step 3, and before issuing all the required commands in step 4, the restore operation can be aborted by issuing:  

```
db2 restore db mydb incremental abort
```
3. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.
4. If either restore command fails in step 4, the failing command can be reissued to continue the restore process.

**Example 5**

Following is a typical automatic incremental redirected restore scenario for the same database:

1. Issue a RESTORE DATABASE command with the INCREMENTAL AUTOMATIC and REDIRECT options.  

```
db2 restore db mydb incremental automatic taken at <ts2>
replace existing redirect
```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

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

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

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

```
db2 restore db mydb continue
```

This is the final step of the redirected restore operation.

**Note:**

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

```
db2 restore db mydb abort
```

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

```
db2 restore db mydb incremental abort
```

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

When you restore a database, the restore utility assumes that the physical container layout will be identical to that of the database when it was backed up.

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

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

The generated script consists of four sections:

### Initialization

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

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

where

- S ON specifies that execution of the command should stop if a command error occurs
- Z ON SAMPLE\_NODE0000.out specifies that output should be directed to a file named *dbalias\_NODEdbpartitionnum.out*
- V ON specifies that the current command should be printed to standard output.

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

### RESTORE DATABASE command with the REDIRECT parameter

The second section starts the **RESTORE DATABASE** command and uses the **REDIRECT** parameter. This section can use all of the **RESTORE DATABASE**

command parameters, except any parameters that cannot be used with the **REDIRECT** parameter. The following is an example of the second section:

```
RESTORE DATABASE SAMPLE
-- USER 'username'
-- USING 'password'
FROM '/home/jseifert/backups'
TAKEN AT 20050906194027
-- DBPATH ON 'target-directory'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/LOGSTREAM0000/'
-- WITH num-buff BUFFERS
-- BUFFER buffer-size
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM n
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
```

### Table space definitions

This section contains table space definitions for each table space in the backup image or specified on the command line. There is a section for each table space, consisting of a comment block that contains information about the name, type and size of the table space. The information is provided in the same format as a table space snapshot. You can use the information provided to determine the required size for the table space. In cases where you are viewing output of a table space created using automatic storage, you will not see a SET TABLESPACE CONTAINERS clause. The following is an example of the table space definition section:

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

```

-- ** Tablespace Content Type                = Any data
-- ** Tablespace Page size (bytes)           = 4096
-- ** Tablespace Extent size (pages)         = 32
-- ** Using automatic storage                = No
-- ** Auto-resize enabled                    = No
-- ** Total number of pages                  = 2000
-- ** Number of usable pages                 = 1960
-- ** High water mark (pages)                = 96
-- *****
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
    FILE    '/tmp/dms1'                      1000
    , FILE  '/tmp/dms2'                      1000
);

```

### RESTORE DATABASE command with the CONTINUE parameter

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

```
RESTORE DATABASE SAMPLE CONTINUE;
```

## Performing a redirected restore using an automatically generated script

When you perform a redirected restore operation, you must specify the locations of physical containers that are stored in the backup image and provide the complete set of containers for each table space that you are altering.

### Before you begin

You can perform a redirected restore only if the database was previously backed up using the Db2 backup utility.

### About this task

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

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

- Even if you specify the **REPLACE EXISTING** parameter when you issue the **RESTORE DATABASE** command to generate the script, the **REPLACE EXISTING** parameter is commented out in the script.
- For security reasons, your password does not appear in the generated script. You need to enter the password manually.
- The restore script includes the storage group associations for every table space that you restore.

### Procedure

To perform a redirected restore using a script:

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

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

This creates a redirected restore script on the client called test\_node0000.clp.

2. Open the redirected restore script in a text editor to make any modifications that are required. You can modify:

- Restore options
- Automatic storage paths
- Container layout and paths

3. Run the modified redirected restore script. For example:

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

## Cloning a production database using different storage group paths

You might have to clone a production database onto a test database that uses a different machine. The test machine and production server are likely to have different storage group paths. The test system might not have paths backed by the newest and fastest storage disks.

### About this task

Suppose you have a production database proddb, where some data is in storage group sg\_hot, which has paths on an SSD device. You want to restore the data into the less expensive and lower-performance test database testdb. The test system does not have the SSD device, but the other paths are equivalent. Performing a redirected restore can change the paths for sg\_hot on the test system without changing the other storage groups.

### Procedure

To restore data from a production database to a test database:

1. Back up the production database. Issue the following command:

```
BACKUP DATABASE production_db TO /backup
```

where *production\_db* is the production database.

2. Set up a redirected restore to the test database. Issue the following command:

```
RESTORE DATABASE testdb REDIRECT
```

where *testdb* is the test database.

3. Modify the storage paths for sg\_hot because no hot storage is available on the test database. Issue the following command:

```
SET STOGROUP PATHS FOR sg_hot ON '/hdd/path1', '/hdd/path2'
```

where *sg\_hot* is the sg\_hot storage group.

4. Proceed with the test database restore. Issue the following command:

```
RESTORE DATABASE testdb CONTINUE
```

5. Update the storage group name to correspond with the new paths. Use the following commands:



```
CONNECT TO testdb
RENAME STOGROUP sg_hot TO sg_cold
```

## Database rebuild

Rebuilding a database is the process of restoring a database or a subset of its table spaces using a set of restore operations. The functionality provided with database rebuild makes Db2 database products more robust and versatile, and provides you with a more complete recovery solution.

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

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

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

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

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

## Specifying the target image

To perform a rebuild of a database, you start by issuing the **RESTORE** command, specifying the most recent backup image that you use as the target of the restore operation. This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. The rebuild target image is specified using the **TAKEN AT** parameter in the **RESTORE DATABASE** command. The target image can be any type of backup (full, table space, incremental, online or offline). The table spaces defined in the database at the time the target image was created will be the table spaces available to rebuild the database.

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

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

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

## Rebuild phase

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

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

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

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

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

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

Once the target image is restored:

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

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

## Automatic processing

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

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

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

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

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

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

## Completing the rebuild phase

Once all the intended table spaces have been restored you have two options based on the configuration of the database. If the database is nonrecoverable, the database will be connectable and any table spaces restored will be online. Any table spaces that are in drop pending state can no longer be recovered and should be dropped if future backups will be performed on the database.

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

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

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

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

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

The correct log files must be available. If you rely on the rollforward utility to retrieve the logs, you must ensure that the Db2 Log Manager is configured to indicate the location from which log files can be retrieved. If the log path or archive path has changed, you need to use the **OVERFLOW LOG PATH** option of the **ROLLFORWARD DATABASE** command.

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

If the error cannot be fixed, you will be able to bring the database up at the point of the failure by issuing the **ROLLFORWARD STOP** command. Any log data beyond that point in the logs will no longer be available once the **STOP** option is used. The database comes up at that point and any table spaces that have been recovered are online. Table spaces that have not yet been recovered are in restore pending state. The database is in the normal state.

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

### Database rebuild and table space containers

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

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

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

### Rebuild with redirected restore

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

If the data for a table space that is restored cannot fit into the new container definitions, the table space is put into restore-pending state and a warning (SQL2563W) is returned to you at the end of the restore. There will also be a message in the Db2 diagnostics log detailing the problem.

### Database rebuild and temporary table spaces

Temporary table spaces are stored differently than other database components in a backup image. Because they are stored differently, temporary table spaces are rebuilt differently during a database restoration.

In general, a Db2 backup image is made up of the following components:

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

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

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

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

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

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

### **Choosing a target image for database rebuild**

The rebuild target image should be the most recent backup image that you want to use as the starting point of your restore operation.

This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. It can be any type of backup (full, table space, incremental, online or offline).

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

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

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

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

Figure 23 on page 395 shows that the following database-level backups and table space-level backups that have been taken, in chronological order:

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

4. Full table space backup TS3
5. Database restore and roll forward to a point between TS1 and TS2
6. Full table space backup TS4
7. Full table space backup TS5

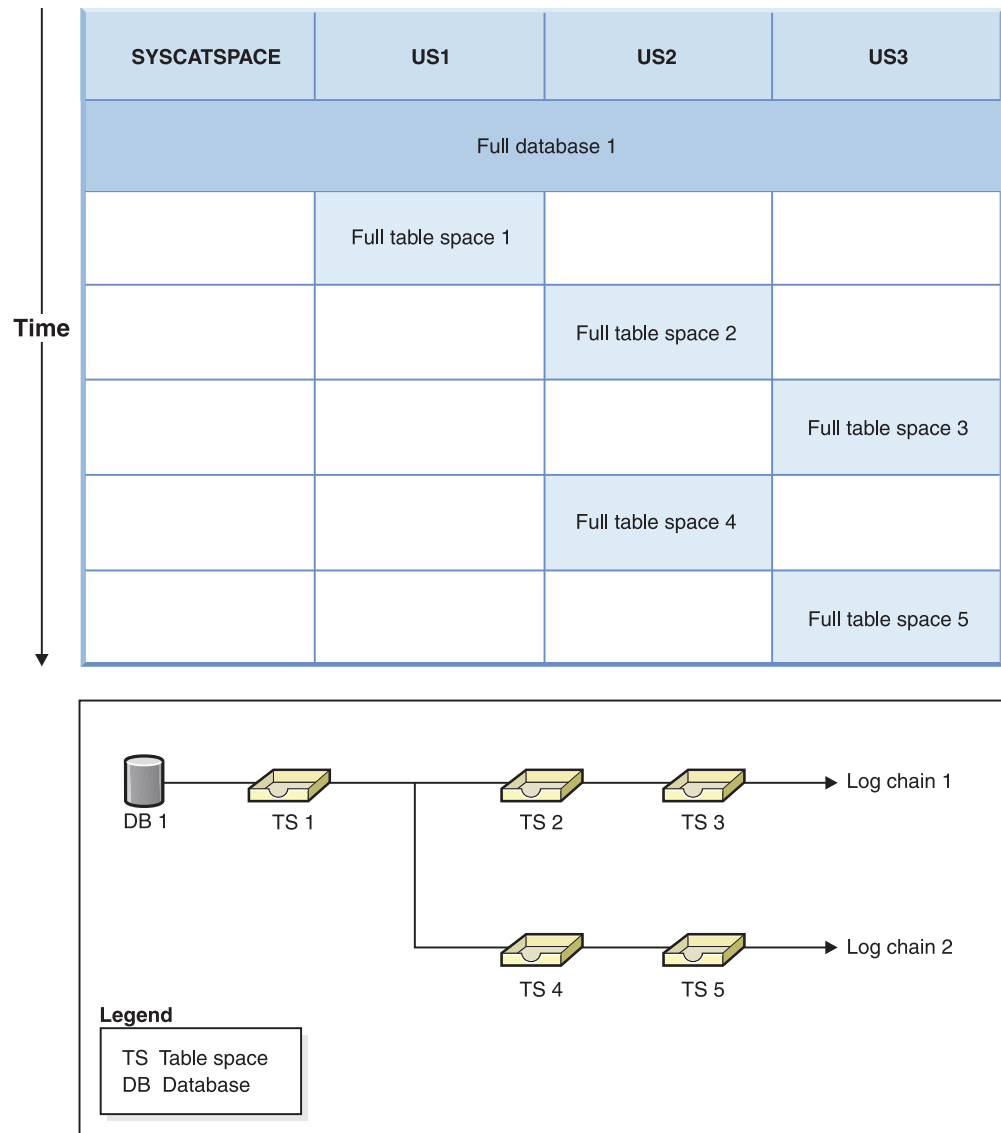


Figure 23. Database and table space-level backups of database SAMPLE

### Example 1

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

```

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

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

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

## Example 2

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

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

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

### Note:

- All logs belonging to log chain 1 must be accessible for the rollforward operations to complete.
- This command may fail because a log file is retrieved from a higher log chain (the rollforward utility always attempts to get log files from the highest log chain), you need to do the following steps:
  1. Extract the log files manually to the overflow log path.
  2. Run the **ROLLFORWARD** command. Include the parameters **-OVERFLOW LOG PATH**, to specify the location of the extracted log files, and **-NORETRIEVE**, to disable the retrieval of archived logs.

## Choosing the wrong target image for rebuild

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

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

Figure 24 on page 397 shows the backup log chain for SAMPLE2, which consists of the following backups:

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



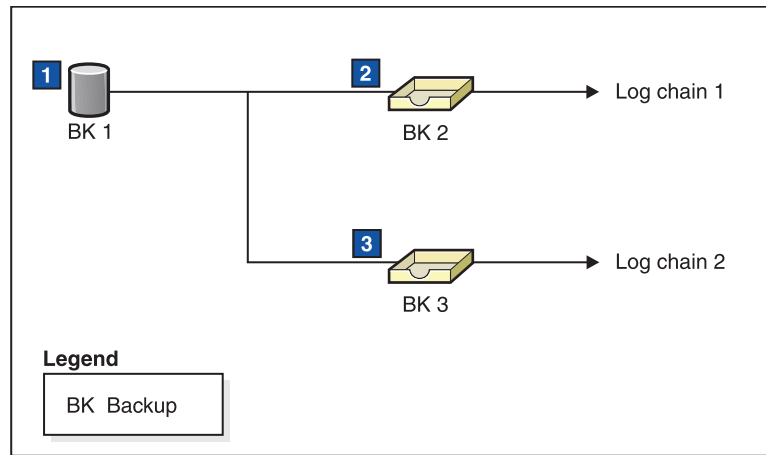


Figure 24. Backup log chain for database SAMPLE2

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

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

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

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

This restore fails and provides a message that says BK2 is from the wrong log chain (SQL2154N). As you can see in Figure 24, the only image that can be used to restore USERSP1 is BK1. Therefore, you need to type the following command:

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

This succeeds so that database can be rolled forward accordingly.

## Rebuilding selected table spaces

Rebuilding a database allows you to build a database that contains a subset of the table spaces that make up the original database.

### About this task

Rebuilding only a subset of table spaces within a database can be useful in the following situations:

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

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

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

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

- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 25 shows that the following backups have been taken:

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

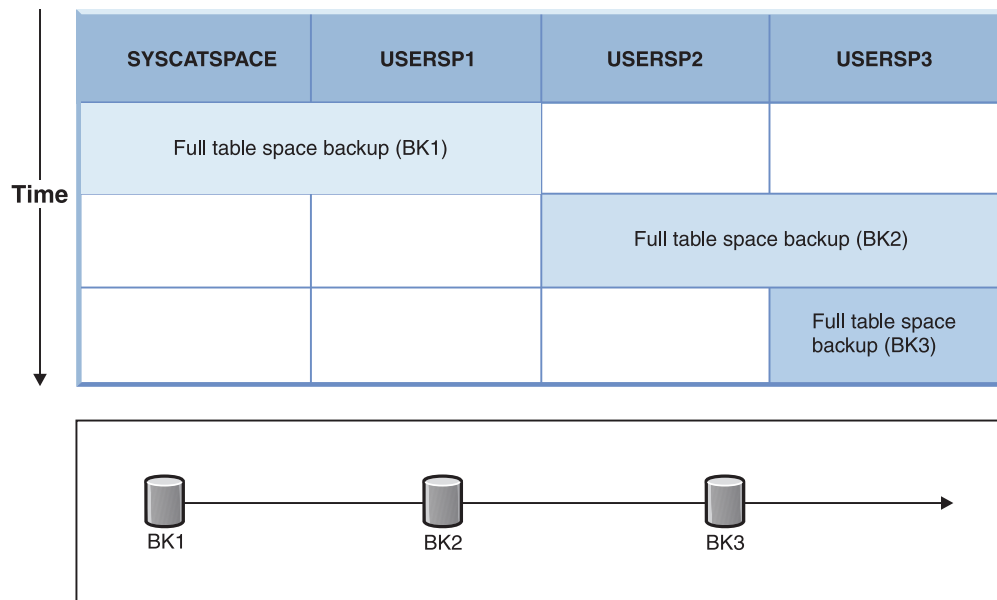


Figure 25. Backup images available for database SAMPLE

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

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

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

## Rebuild and incremental backup images

You can rebuild a database using incremental images.

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

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

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

## Rebuilding partitioned databases

To rebuild a partitioned database, rebuild each database partition separately. For each database partition, beginning with the catalog partition, first restore all the table spaces that you require. Any table spaces that are not restored are placed in restore pending state.

Once all the database partitions are restored, you then issue the **ROLLFORWARD DATABASE** command on the catalog partition to roll all of the database partitions forward.

### About this task

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

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

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

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

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

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

- BK43 is a backup of USERSP3

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

### Procedure

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

```
db2 restore db sample rebuild with all tablespaces in database
taken at BK31 without prompting
```
2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:  

```
db2 restore db sample rebuild with tablespaces in database
taken at BK42 without prompting
```
3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:  

```
db2 restore db sample rebuild with all tablespaces in database
taken at BK43 without prompting
```
4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:  

```
db2 rollforward db sample to end of logs
```
5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:  

```
db2 rollforward db sample stop
```

### What to do next

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

### Restrictions for database rebuild

You can use the REBUILD option to complete a set of restore commands, but it has restrictions that you need to be aware of.

The following list is a summary of database rebuild restrictions:

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

## Rebuild sessions - CLP examples

This topic provides a number of examples of rebuild operations.

### Scenario 1

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

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

The following backups have been taken:

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

#### Example 1

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

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

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK3 without prompting
```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):  

```
db2 rollforward db mydb to end of logs
```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:  

```
db2 rollforward db mydb stop
```

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

#### Example 2

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

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

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)  
taken at BK2 without prompting
```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO PIT** option (this assumes all logs have been saved and are accessible):  

```
db2 rollforward db mydb to PIT
```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:  

```
db2 rollforward db mydb stop
```

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

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

1. Issue the **RESTORE DATABASE** command *without* the **REBUILD** option and specify the table space you want to restore. First restore USERSP1:

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

2. Then restore USERSP3:

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

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

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

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

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

```
db2 rollforward db mydb stop
```

### Example 3

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

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

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

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

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

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

```
db2 rollforward db mydb stop
```

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

### Example 4

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

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

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

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

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

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

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

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

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

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

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

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

```
db2 rollforward db mydb stop
```

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

### Example 5

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

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

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

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

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

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

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

```
db2 rollforward db newdb stop
```

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

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

## Scenario 2

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

### Example 6

The following restores all table spaces except T0999 and T1000:

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

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

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

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

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

```
db2 rollforward db mydb stop
```

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

### Scenario 3

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

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

The following backups have been taken:

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

#### Example 7

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

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

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

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

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

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

```
db2 rollforward db mydb stop
```

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

#### Example 8

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

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

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



2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):  
`db2 rollforward db mydb to end of logs`
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:  
`db2 rollforward db mydb stop`

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

#### Example 9

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

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option. Although the target image is a non-incremental image, the restore utility will detect that the required rebuild chain includes incremental images and it will automatically restore those images incrementally.  
`db2 restore db mydb rebuild with all tablespaces in database except  
tablespace (USERSP3) taken at FULL2 without prompting`
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):  
`db2 rollforward db mydb to end of logs`
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:  
`db2 rollforward db mydb stop`

#### Scenario 4

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

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

#### Example 10

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

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:  
`db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)  
taken at BK1 logtarget /logs without prompting`
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs after the end of BK1 have been saved and are accessible):  
`db2 rollforward db mydb to end of logs overflow log path (/logs)`
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:  
`db2 rollforward db mydb stop`

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

#### Example 11

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

- BK1 is a backup of SYSCATSPACE, using log files 10-45

- BK2 is a backup of USERSP1 and USERSP2, using log files 64-80

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

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

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

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

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

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

```
db2 rollforward db mydb stop
```

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

## Scenario 5

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

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

The following backups have been taken:

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

### Example 12

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

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

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

2. Issue a **SET TABLESPACE CONTAINERS** command for each table space whose containers you want to redefine. For example:

```
db2 set tablespace containers for 3 using (file '/newusersp1' 10000)
```

- 3.

```
db2 set tablespace containers for 4 using (file '/newusersp2' 15000)
```

4. Issue a **RESTORE DATABASE** command with the **CONTINUE** option:

```
db2 restore db mydb continue
```

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

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

6. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

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

## Scenario 6

In the following examples, there is a database called MYDB with three database partitions:

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

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

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

### Example 13

The following rebuilds the entire database to the end of logs.

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

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

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

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

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

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

4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (assumes all logs have been saved and are accessible on all database partitions):

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

5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

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

### Example 14

The following rebuilds SYSCATSPACE, USERSP1 and USERSP2 to the most recent point in time.

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

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

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

```
db2 restore db mydb rebuild with all tablespaces in image taken at
BK22 without prompting
```

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

```
db2 restore db mydb rebuild with all tablespaces in image taken at
BK33 without prompting
```

Note: this command omitted **USERSP1**, which is needed to complete the rebuild operation.

4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:

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

5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

The rollforward succeeds and the database is connectable on all database partitions. All table spaces are in **NORMAL** state, except **USERSP3**, which is in **RESTORE PENDING** state on all database partitions on which it exists, and **USERSP1**, which is in **RESTORE PENDING** state on database partition 3.

When an attempt is made to access data in **USERSP1** on database partition 3, a data access error will occur. To fix this, **USERSP1** will need to be recovered:

- a. On database partitions 3, issue a **RESTORE DATABASE** command, specifying a backup image that contains **USERSP1**:

```
db2 restore db mydb tablespace taken at BK23 without prompting
```

- b. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option and the **AND STOP** option:

```
db2 rollforward db mydb to end of logs on dbpartitionnum (3) and stop
```

At this point **USERSP1** on database partition 3 can have its data accessed since it is in **NORMAL** state.

## Scenario 7

In the following examples, there is a *nonrecoverable* database called **MYDB** with the following table spaces:

- **SYSCATSPACE** (0), SMS system catalog
- **USERSP1** (1) DMS user data table space
- **USERSP2** (2) DMS user data table space

There is just one backup of the database, **BK1**:

### Example 15

The following demonstrates using rebuild on a nonrecoverable database.

Rebuild the database using only **SYSCATSPACE** and **USERSP1**:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1)
taken at BK1 without prompting
```

Following the restore, the database is connectable. If you issue the **LIST TABLESPACES** command or the **MON\_GET\_TABLESPACE** table function, you see that the SYSCATSPACE and USERSP1 are in NORMAL state, while USERSP2 is in DELETE\_PENDING/OFFLINE state. You can now work with the two table spaces that are in NORMAL state.

If you want to do a database backup, you will first need to drop USERSP2 using the **DROP TABLESPACE** statement, otherwise, the backup will fail.

To restore USERSP2 at a later time, you need to reissue a database restore from BK1.

## Monitoring the progress of restore operations

You can use the **LIST UTILITIES** command to monitor restore operations on a database.

### Procedure

Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter

```
LIST UTILITIES SHOW DETAIL
```

### Results

For restore operations, an initial estimate is not given. Instead, UNKNOWN is specified. As each buffer is read from the image, the actual number of bytes read is updated. For automatic incremental restore operations where multiple images might be restored, the progress is tracked by using phases. Each phase represents an image to be restored from the incremental chain. Initially, only one phase is indicated. After the first image is restored, the total number of phases will be indicated. As each image is restored the number of phases completed is updated, as is the number of bytes processed.

### Example

The following is an example of the output for monitoring the performance of a restore operation:

|                      |                              |
|----------------------|------------------------------|
| ID                   | = 6                          |
| Type                 | = RESTORE                    |
| Database Name        | = SAMPLE                     |
| Partition Number     | = 0                          |
| Description          | = db                         |
| Start Time           | = 08/04/2011 12:24:47.494191 |
| State                | = Executing                  |
| Invocation Type      | = User                       |
| Progress Monitoring: |                              |
| Completed Work       | = 4096 bytes                 |
| Start Time           | = 08/04/2011 12:24:47.494197 |

## Optimizing restore performance

When you perform a restore operation, Db2 database products will automatically choose an optimal value for the number of buffers, the buffersize and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration.

Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the **util\_heap\_sz**

configuration parameter. The objective is to minimize the time it takes to complete a restore operation. Unless you explicitly enter a value for the following **RESTORE DATABASE** command parameters, Db2 database products will select one for them:

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

For restore operations, a multiple of the buffer size used by the backup operation will always be used. You can specify a buffer size when you issue the **RESTORE DATABASE** command but you need to make sure that it is a multiple of the backup buffer size.

You can also choose to do any of the following to reduce the amount of time required to complete a restore operation:

- Increase the restore buffer size.  
The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers allocated will be the smallest acceptable size.
- Increase the number of buffers.  
The value you specify must be a multiple of the buffersize that was used for the backup, otherwise it will be rounded down to the closest multiple of the backup buffersize.
- Increase the value of the **PARALLELISM** parameter.  
This will increase the number of buffer manipulators (BM) that will be used to write to the database during the restore operation.
- Increase the utility heap size  
This will increase the memory that can be used simultaneously by the other utilities.

## Privileges, authorities, and authorization required to use restore

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.

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.

## Database schema transporting

Transporting a database schema involves taking a backup image of a database and restoring the database schema to a different, existing database.

When you transport a database schema, the database objects in the transported schema are re-created to reference the new database, and the data is restored to the new database.

A database schema must be transported in its entirety. If a table space contains both the schema you want to transport, as well as another schema, you must transport all data objects from both schemas. These sets of schemas that have no references to other database schemas are called *transportable sets*. The data in the table spaces and logical objects in the schemas in a transportable set reference only table spaces and schemas in the transportable set. For example, tables have table dependencies only on other tables in the transportable set.

The following diagram illustrates a database with several table spaces and schemas. In the diagram, the table spaces that are referenced by the schemas are above the schemas. Some schemas reference multiple table spaces and some table spaces are referenced by multiple schemas.

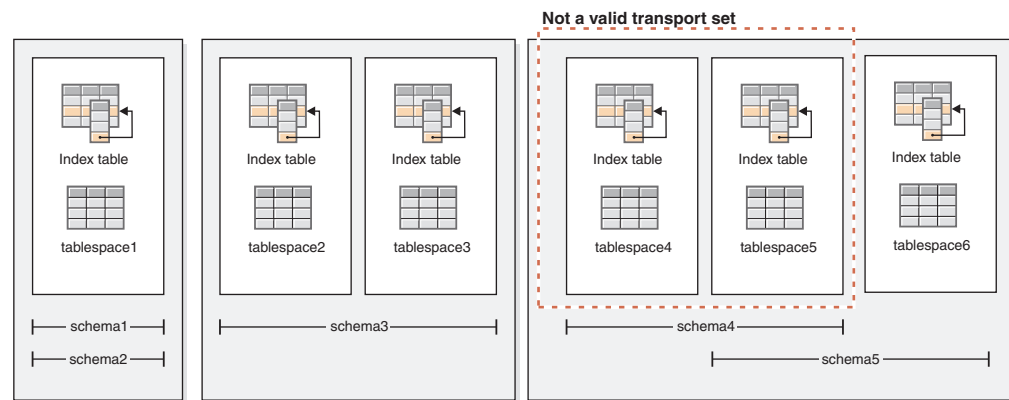


Figure 26. Sets of table spaces and schemas

The following combinations of table spaces and schemas are valid transportable sets:

- tablespace1 with schema1 and schema2
- tablespace2 and tablespace3 with schema3
- tablespace4, tablespace5, and tablespace6, with schema4 and schema5
- A combination of valid transportable sets also constitutes a valid transportable set:
  - tablespace1, tablespace2, and tablespace3, with schema1, schema2, and schema3

The set tablespace4 and tablespace5 with schema4 is not a valid transportable set because there are references between tablespace5 and schema5 and between schema5 and tablespace6. The set requires tablespace6 with schema5 to be a valid transportable set.

You can transport database schemas by using the **RESTORE** command with the **TRANSPORT** parameter.

**Note:** The **TRANSPORT** option is not supported in the Db2 pureScale environment, or in partitioned database environments.

When you transport database schemas, a temporary database is created and named as a part of the transport operation. This *transport staging database* is used to extract logical objects from the backup image so that they can be re-created on the target database. If logs are included in the backup image, they are also used to bring the staging database to a point of transactional consistency. The ownership of the transported table spaces is then transferred to the target database.

## Considerations about the database objects re-created when transporting database schemas

Review the following information related to the re-creation of database objects when you are transporting database schemas:

*Table 20. Transport considerations for specific database objects*

| Database object                                 | Consideration when transporting schemas                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQL routines (not external routines using SQL)  | A new copy of the SQL routine is created in the target database. For SQL stored procedures, additional catalog space is consumed because an additional copy of the stored procedure byte code is created in the new database.                                                                                                                                                                                                                                                                                                                                      |
| External routines                               | A new catalog entry is created for each routine. This catalog entry references the same binary file as the original source routine. The <b>RESTORE</b> command does not copy the external routine binary file from the source system.                                                                                                                                                                                                                                                                                                                              |
| Source tables in states causing access problems | For tables that are not in normal state at the time the backup image was generated, such as tables in check pending state or load pending state, the data from those tables might not be accessible in the target database. To avoid having this inaccessible data, you can move the tables to normal state in the source database before schema transport.                                                                                                                                                                                                        |
| Tables containing the data capture attribute    | Source tables with data capture enabled are transported to the target database with the data capture attribute and continue to log interdatabase data replication information. However, replicated tables do not extract information from this table. You have the option of registering the new target table to act as a replication source after the <b>RESTORE</b> command completes.                                                                                                                                                                           |
| Tables using label-based access control (LBAC)  | When transporting data that is protected by LBAC, the transport operation re-creates the LBAC objects on the target database. If LBAC objects of the same name exist on the target database, the transport operation fails. To ensure that restricted data access is not compromised, the transport operation does not use existing LBAC objects on the target database.                                                                                                                                                                                           |
| Temporary table spaces                          | If there are any system temporary table spaces that are defined with the source backup image and the transport operation excludes them from the table space list, these system temporary table spaces are still created in the staging database but not the final target database. As a result, you must issue the <b>SET TABLESPACE CONTAINERS</b> command for these system temporary table spaces in order to provide valid containers to complete the restore operation, just as you would for any table spaces that are specified within the table space list. |

When you transport table spaces, a log record with a special format is created on the target database. This format cannot be read by previous Db2 versions. If you transport table spaces and then downgrade to a version earlier than Db2 Version 9.7 Fix Pack 2, then you cannot recover the target database that contains the table spaces that were transported. To ensure that the target database is compatible with earlier Db2 versions, you can roll forward the target database to a point in time before the transport operation.

**Important:** If database rollforward detects a table space schema transport log record, the corresponding transported table space is taken offline and moved into drop pending state. This is because database does not have complete logs of transported table spaces to rebuild transported table spaces and their contents. You



can take a full backup of the target database after transport completes, so subsequent rollforward does not pass the point of schema transport in the log stream.

## Transportable objects

When you transport data from a backup image to a target database, there are two main results. The physical and logical objects in the table spaces that you are restoring are re-created in the target database, and the table space definitions and containers are added to the target database.

The following logical objects are re-created:

- Tables, created global temporary tables, and materialized query tables
- Normal and statistical views
- The following types of generated columns:
  - Expression
  - Identity
  - Row change timestamp
  - Row change token
- User-defined functions and generated functions
- Functions and procedures except for external routine executables
- User-defined types
- The following types of constraints:
  - Check
  - Foreign key
  - Functional dependency
  - Primary
  - Unique
- Indexes
- Triggers
- Sequences
- Object authorizations, privileges, security, access control, and audit configuration
- Table statistics, profiles, and hints
- Packages

The following components of a schema are not created on the target database:

- Aliases
- Column-organized tables
- Created global variables
- External routine executable files
- Functional mappings and templates
- Hierarchy tables
- Index extensions
- Jobs
- Methods
- Nicknames
- OLE DB external functions
- Range-partitioned tables

- Servers
- Sourced procedures
- Structured types
- System catalogs
- Typed tables and typed views
- Usage lists
- Wrappers

**Important:** Expression based indexes are not supported when using the schema transport feature.

## Transport examples

You can use the **RESTORE DATABASE** command with the **TRANSPORT** option to copy a set of table spaces and SQL schemas from one database to another database.

The following examples use a database named **ORIGINALDB** as source of the backup image and the target database **TARGETDB**.

The following illustration shows the **ORIGINALDB** table spaces and schemas:

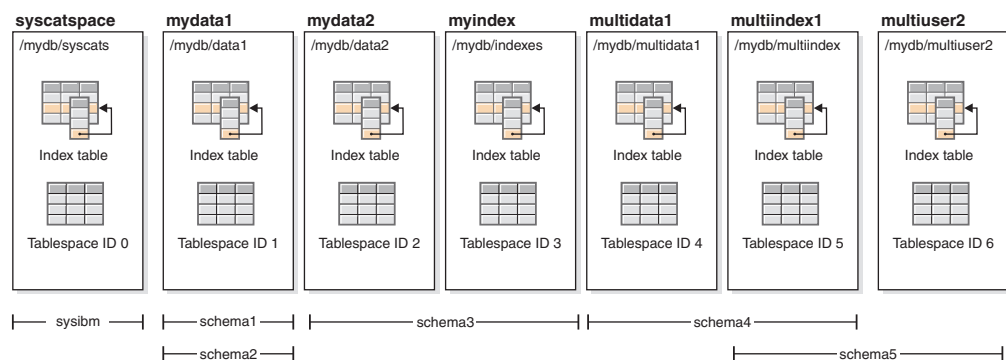


Figure 27. ORIGINALDB database

The originalDB database contains the following valid transportable sets:

- mydata1; schema1 + schema2
- mydata2 + myindex; schema3
- multidata1 + multiindex1 + multiuser2; schema4 + schema5
- A combination of valid transportable sets also constitutes a valid transportable set:
  - mydata1 + mydata2 + myindex; schema1 + schema + schema3

The following illustration shows the **TARGETDB** table spaces and schemas:

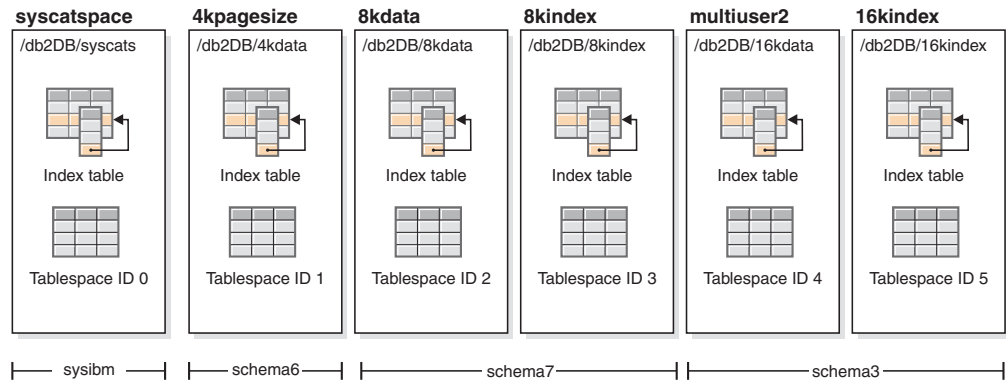


Figure 28. TARGETDB database

If the sources and target databases contain any schemas with the same schema name, or any table spaces of the table space name, then you cannot transport that schema or table space to the target database. Issuing a transport operation that contains a schema or a table space that has the same name as a schema or a table space on the target database will cause the transport operation to fail. For example, even though the following grouping is a valid transportable set, it cannot be directly transported to the target database:

- mydata2 + myindex; schema3 (schema3 exists in both the source and target databases)

If there exists a single online backup image for ORIGINALDB that contains all of the table spaces in the database, then this will be the source for the transport. This also applies to table space level backup images.

You can redirect the container paths for the table spaces being transported. This is especially important if database relative paths were used.

## Examples

*Example 1:* Successfully transport the schemas schema1 and schema2 in the mydata1 table space into TARGETDB.

```
db2 restore db originaldb tablespace (mydata1) schema(schema1,schema2)
from <Media_Target_clause> taken at <date-time>
transport into targetdb redirect
```

```
db2 list tablespaces
db2 set tablespace containers for <tablespace ID for mydata1>
using (path '/db2DB/data1')
```

```
db2 restore db originaldb continue
```

The resulting TARGETDB will contain the mydata1 table space and schema1 and schema2.

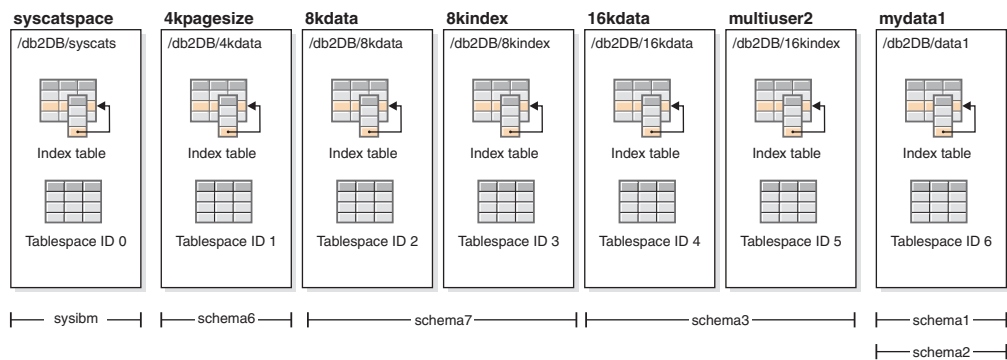


Figure 29. TARGETDB database after transport

*Example 2:* Transport the schema schema3 in the mydata2 and myindex table spaces into TARGETDB. You cannot transport a schema that already exists on the target database.

```
db2 restore db originaldb tablespace (mydata2,myindex) schema(schema3)
transport into targetdb
```

The transport operation will fail because the schema schema3 already exists on the target database. TARGETDB will remain unchanged. SQLCODE=SQL2590N rc=3.

*Example 3:* Transport the schemas schema4 and schema5 in the multidata1, multiindex1, and multiuser2 table spaces into TARGETDB. You cannot transport a table space that already exists on the target database.

```
db2 restore db originaldb tablespace (multidata1,multiindex1,multiuser2)
schema(schema4,schema5) transport into targetdb
```

The transport operation will fail and TARGETDB will remain unchanged because table space multiuser2 already exists on the target database. SQLCODE=SQL2590N rc=3.

*Example 4:* Transport the myindex table space into TARGETDB. You cannot transport partial schemas.

```
db2 restore db originaldb tablespace (myindex) schema(schema3)
transport into targetdb
```

The list of table spaces and schemas being transported is not a valid transportable set. The transport operation will fail and TARGETDB will remain unchanged. SQLCODE=SQL2590N rc=1.

*Example 5:* Restore the syscatspace table space into TARGETDB. You cannot transport system catalogs.

```
db2 restore db originaldb tablespace (syscatspace) schema(sysibm)
transport into targetdb
```

The transport operation will fail because the system catalogs can not be transported. SQLCODE=SQL2590N rc=4. You can transport user defined table spaces or restore the system catalogs with the RESTORE DATABASE command without specifying the transport option.

*Example 6:* You cannot restore into a target database that does not exist on the system.

```
db2 restore db originaldb tablespace (mydata1) schema(schema1,schema2)
transport into notexists
```

The transport operation will fail. Table spaces cannot be transported to a target database that does not exist.

### Troubleshooting: transporting schemas

If an error occurs on either the staging or target database, you must redo the entire restore operation. All failures that occur are logged in the `db2diag` log file on the target server. Review the **db2diag** log before reissuing the **RESTORE** command.

### Dealing with errors

Errors occurring during restore are handled in various ways depending on the type of object being copied and the phase of transport. There might be circumstances, such as a power failure, in which not everything is cleaned up.

The transport operation consists of the following phases:

- Staging database creation
- Physical table space container restoration
- Rollforward processing
- Schema validation
- Transfer of ownership of the table space containers
- Schema re-creation in target database
- Dropping the staging database (if the **STAGE IN** parameter is not specified)

If any errors are logged at the end of the schema re-creation phase, about transporting physical objects, then the restore operation fails and an error is returned. All object creation on the target database is rolled back, and all internally created tables are cleaned up on the staging database. The rollback occurs at the end of the re-create phase, to allow all possible errors to be recorded into the **db2diag** log file. You can investigate all errors returned before reissuing the command.

The staging database is dropped automatically after success or failure. However, it is not dropped in the event of failure if the **STAGE IN** parameter is specified. The staging database must be dropped before the staging database name can be reused.

## Restore from Db2 pureScale Feature to Db2 Enterprise Server Edition

Restore of offline backup images that are taken on Db2 pureScale instance to Db2 Enterprise Server Edition , without roll forward support through the transition.

### About this task

Consider the following before you restore a database from a Db2 pureScale instance to a Db2 Enterprise Server Edition instance:

- Only backups of consistent databases are supported.
- To be able to support recoverability from the point in time of the restore operation, after completing the restore operation, you must take a new offline database backup.
- The source and target instances must be from the same Db2 product level.
- The target member topology must include the member identifier of the Db2 ESE instance.

## Restrictions

If you are restoring a backup image of an inconsistent database from a Db2 pureScale Feature to a Db2 Enterprise Server Edition instance, as a workaround, you can first perform a full database backup on the source instance, then restore the full database backup by running the **RESTORE** command.

## Procedure

To restore a database from a Db2 pureScale instance to a Db2 Enterprise Server Edition instance:

1. On the source instance (Db2 pureScale instance with, for example, members 0 and 1), perform an offline database backup:  
`db2 backup database sample to /dev3/backup`
2. On the target instance (ESE, member 0), perform a restore:  
`db2 restore database sample from /dev3/backup without rolling forward`
3. Perform an offline database backup:  
`db2 backup database sample to /dev3/backup`

## Results

The database is now usable.

## Restore from Db2 Enterprise Server Edition to Db2 pureScale instance

You can easily restore offline backup images that are taken on ESE to a Db2 pureScale instance.

### Before you begin

Before you can restore a Db2 Enterprise Server Edition database backup to a Db2 pureScale instance, you can verify that your databases are ready for use in a Db2 pureScale environment. To verify that your ESE database is ready to restore to a Db2 pureScale environment, run the **db2checkSD** command.

### About this task

These requirements must be considered before you restore a database from a Db2 Enterprise Server Edition instance to a Db2 pureScale instance:

- Only backups of consistent databases are supported.
- To be able to support recoverability from the point in time of the restore operation, after you complete the restore operation, you must take a new offline full database backup.
- The source and target instances must be from the same Db2 product level.
- The target member topology must include the member identifier of the Db2 ESE instance.

## Restrictions

If you are restoring a backup image of an inconsistent database between a Db2 Enterprise Server Edition instance and a Db2 pureScale instance, as a workaround, you can either perform an offline backup on the source instance or update the

target instance to include a member identifier in the source instance, then rerun the **RESTORE** command.

## Procedure

To restore a database from a Db2 Enterprise Server Edition instance to a Db2 pureScale instance:

1. Optional: On the source instance (ESE), run the **db2checkSD** command:

```
db2checkSD dbname -l filename -u userid -p password
```

At this point, the **db2checkSD** command performs compatibility checks that determine if the database can be used in a Db2 pureScale environment.

2. Perform an offline database backup:

```
db2 backup database dbname to directory
```

3. On the target instance (Db2 pureScale instance that includes, for example, member 0 in the topology), restore the backup on the Db2 pureScale instance with members 0, 1, and 2 from member 0 (the common member).

```
db2 restore database dbname from directory without rolling forward
```

4. On the target instance, run the **db2checkSD** command again:

```
db2checkSD dbname -l filename -u userid -p password
```

At this point, the **db2checkSD** command performs the conversion of the database so that it can be used in a Db2 pureScale environment. The user ID must be the same as the user ID that has DATAACCESS authority on the source instance.

5. Optional: Perform either an incremental or full offline database backup from member 0 (the common member) by issuing the following command:

```
db2 backup database dbname to directory
```

## Results

The database is now usable.

---

## Rollforward overview

You cannot recover transactions that occurred after backup image creation by using the restore tools. Instead, you can recover transactions that have occurred since the last backup command was completed by using rollforward commands. You must enable database logging for these commands to be effective.

The simplest form of the **ROLLFORWARD DATABASE** command requires only that you specify the alias name of the database that you want to rollforward recover, as in the following example:

```
db2 ROLLFORWARD DB sample
```

In IBM Data Studio Version 3.1 or later, you can use the task assistant for rolling forward databases. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

The following is one approach you can use to perform rollforward recovery:

1. Invoke the rollforward utility without the **STOP** option.

2. Invoke the rollforward utility with the **QUERY STATUS** option  
If you specify recovery to the end of the logs, the **QUERY STATUS** option can indicate that one or more log files are missing, if the returned point in time is earlier than you expect.  
If you specify point-in-time recovery, the **QUERY STATUS** option helps you to ensure that the rollforward operation completes at the correct point.
3. Invoke the rollforward utility with the **STOP** option. After the operation stops, it is not possible to roll additional changes forward.

An alternate approach you can use to perform rollforward recovery is the following:

1. Invoke the rollforward utility with the **AND STOP** option.
2. The need to take further steps depends on the outcome of the rollforward operation:
  - If it is successful, the rollforward is complete and the database is connectable and usable. At this point, it is not possible to roll additional changes forward.
  - If any errors were returned, take whatever action is required to fix the problem. For example, if there is a missing log file: find the log file, or if there are retrieve errors: ensure that log archiving is working. Then reissue the rollforward utility with the **AND STOP** option.

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

When the rollforward utility is invoked:

- If the database is in rollforward pending state, the database is rolled forward. Any table spaces that were restored from backup images that were taken after the database backup image, and are currently in rollforward pending state are also rolled forward. Any table spaces that were taken prior to the database level backup and restored after the database level backup was restored remain in rollforward pending state. You must issue a subsequent table space level rollforward to recover them.
- If the database is *not* in rollforward pending state, but table spaces in the database *are* in rollforward pending state:
  - If you specify a list of table spaces, only those table spaces are rolled forward.
  - If you do not specify a list of table spaces, all table spaces that are in rollforward pending state are rolled forward.

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

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

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



When you first create a database, it is enabled for circular logging only. This means that logs are reused, rather than being saved or archived. With circular logging, rollforward recovery is not possible: only crash recovery or version recovery can be done. Archived logs document changes to a database that occur after a backup was taken. You enable log archiving (and rollforward recovery) by setting the **logarchmeth1** database configuration parameter to a value other than its default of OFF. When you set **logarchmeth1** to a value other than OFF, the database is placed in backup pending state, and you must take an offline backup of the database before it can be used again.

**Note:** Entries are made in the recovery history file for each log file that is used in a rollforward operation.  
In this example, the command returns:

In a partitioned database environment and a Db2 pureScale environment, this status information is returned for each database partition or member:

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

```

                                Rollforward Status
Input database alias              = mydb
Number of members have returned status = 3

Member ID Rollforward status  Next log to be read  Log files processed  Last committed transaction
-----
0 DB working S0000001.LOG S0000000.LOG-S0000000.LOG 2009-05-06-15.28.11.000000 UTC
1 DB working S0000010.LOG S0000000.LOG-S0000009.LOG 2009-05-06-15.28.20.000000 UTC
2 DB working S0000005.LOG S0000000.LOG-S0000004.LOG 2009-05-06-15.27.33.000000 UTC

DB20000I The ROLLFORWARD command completed successfully.
```

Using rollforward

Use the **ROLLFORWARD DATABASE** command to apply transactions that were recorded in the database log files to a restored database backup image or table space backup image.

Before you begin

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.

About this task

Do not restore table spaces without canceling a rollforward operation that is in progress. Otherwise, you might have a table space set in which some table spaces are in rollforward in progress state, and some table spaces are in rollforward pending state. A rollforward operation that is in progress only operates on the tables spaces that are in rollforward in progress state.

The database can be local or remote.

The following restrictions apply to the rollforward utility:

- You can invoke only one rollforward operation at a time. If there are many table spaces to recover, you can specify all of them in the same operation.

- If you have renamed a table space following the most recent backup operation, ensure that you use the new name when rolling the table space forward. The previous table space name is not 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** parameter has not been specified, or a rollforward operation that has failed before completing.
- You cannot *continue* a table space rollforward operation to a point in time, specifying a time stamp that is less than the previous one. If a point in time is not specified, the previous one is used. You can issue a rollforward operation that ends at a specified point in time by just specifying **STOP**, but this is only allowed if the table spaces involved were all restored from the same offline backup image. In this case, no log processing is required. If you start another rollforward operation with a different table space list before the in-progress rollforward operation is either completed or cancelled, an error message (SQL4908) is returned. Invoke the **LIST TABLESPACES** command on all database partitions (or use the `MON_GET_TABLESPACE` table function) to determine which table spaces are currently being rolled forward (rollforward in progress state), and which table spaces are ready to be rolled forward (rollforward pending state). You have three options:
  - Finish the in-progress rollforward operation on all table spaces.
  - Finish the in-progress rollforward operation on a subset of table spaces. (This might not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all database partitions.)
  - Cancel the in-progress rollforward operation.
- In a partitioned database environment, the rollforward utility must be invoked from the catalog partition of the database.
- Point in time rollforward of a table space was introduced in Db2 Version 9.1 clients. You should upgrade to Version 11.1 any clients in order to roll a table space forward to a point in time.
- You cannot roll forward logs from a previous release version.

## Procedure

To invoke the rollforward utility, use the:

- **ROLLFORWARD DATABASE** command, or
- db2Rollforward application programming interface (API).
- Open the task assistant in IBM Data Studio for the **ROLLFORWARD DATABASE** command.

## Example

The following is an example of the **ROLLFORWARD DATABASE** command issued through the CLP:

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

## Continuing a stopped or failed rollforward operation

You can continue a rollforward operation if any of the following occurred: the previous rollforward failed; the previous rollforward was interrupted; or the previous rollforward finished, but the command did not specify either **STOP** or **COMPLETE**.

## Before you begin

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.

## About this task

One option you have when continuing a rollforward operation is to use a *forced stop*, which you do by issuing a **ROLLFORWARD DATABASE** command with the **STOP** option, but specifying **T0**. A forced stop means that the rollforward utility ignores certain errors if they are safe to ignore. For example, missing log file errors, checksum errors, log chain errors, and not reaching the point in time are considered ignorable errors. If Db2 determines that it is safe to stop, then the rollforward operation goes through the undo phase, and the database will be available for normal connections. If Db2 determines that it is not safe to stop, then the rollforward operation fails and the database remains in rollforward pending state.

### Restrictions

If you are continuing a rollforward operation that was to a point-in-time, the new rollforward must be one of the following:

- a rollforward to the same point in time
- a rollforward to a later point in time
- a rollforward to the end of logs
- a rollforward with the **STOP** or **COMPLETE** option, but without a point-in-time, **END OF LOGS**, or **END OF BACKUP** option

## Procedure

To continue a rollforward operation, use the following steps:

## Results

### Example

## What to do next

### Rolling forward changes in a table space

If the database is enabled for rollforward recovery, you have the option of backing up, restoring, and rolling forward table spaces instead of the entire database.

You can roll forward changes to a table space independently of other table spaces in your database, or you can roll forward changes to all table spaces at the same time.

Implementing a recovery strategy for individual table spaces can save time because 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, you can restore that table space and roll it 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. (You can restore the system catalog table space

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.

If you want to skip the log files known not to contain any log records affecting the table space, ensure that the **DB2\_COLLECT\_TS\_REC\_INFO** registry variable is set to ON, which is the default setting unless it is a high availability disaster recovery (HADR) database. This registry variable must be set before the log files are created and used so that the information required for skipping log files is collected. If **DB2\_COLLECT\_TS\_REC\_INFO** is set to OFF, all log files are processed even if they do not contain log records that affect that table space when that table space is rolled forward.

**Note:** Table space recovery is not supported on high availability disaster recovery (HADR) databases. As a result, the default setting for the **DB2\_COLLECT\_TS\_REC\_INFO** registry variable is OFF for HADR databases, which avoids the unnecessary overhead of keeping track of which table spaces have changed in each log file.

The table space change history file (DB2TSCHG.HIS), which is located in the database directory, tracks which logs to process for each table space. You can view the contents of this file with the **db2logsForRfwd** utility, and delete entries from it with the **PRUNE HISTORY** command. During a database restore operation, the DB2TSCHG.HIS file is restored from the backup image and then brought up to date during the database rollforward operation. If no information is available for a log file, it is treated as though it is required for the recovery of every table space.

Because information for each log file is flushed to disk after the log becomes inactive, this information can be lost as a result of a crash. To prevent this loss from occurring, if a recovery operation begins in the middle of a log file, the entire log is treated as though it contains modifications to every table space in the system. All active logs are processed and the information for them is rebuilt. If information for older or archived log files is lost in a crash situation and no information for them exists in the data file, they are treated as though they contain modifications for every table space during the table space recovery operation.

Before you roll a table space forward, use the **MON\_GET\_TABLESPACE** table function to determine 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 you are 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 that are being recovered. You cannot roll forward a table space to a time that is earlier than the backup timestamp. In a partitioned database environment, you must roll forward the table spaces to at least the highest minimum recovery time of all the table spaces on all database partitions.

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

If the data and the long objects in a table are in separate table spaces, and the long object data was reorganized, the table spaces for both the data and the long objects must be restored and rolled forward together. Take a backup of the affected table spaces after the table is reorganized.

If you want to roll forward a table space to a point in time, and a table in the table space is either:

- an underlying table for a materialized query or staging table that is in another table space
- a materialized query or staging table for a table in another table space

then roll both table spaces forward to the same point in time. If you do not, the materialized query or staging table is placed in set integrity pending state at the end of the rollforward operation. The materialized query table needs to be fully refreshed, and the staging table is marked as incomplete.

If you want to roll forward a table space 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, roll forward both table spaces simultaneously to the same point in time. If you do not roll forward both table spaces, the child table in the referential integrity relationship is placed in set integrity pending state at the end of the rollforward operation. When the child table is later checked for constraint violations, a check on the entire table is required. If any of the following tables exist, they are also placed in set integrity pending state with the child table:

- any descendant materialized query tables for the child table
- any descendant staging tables for the child table
- any descendant foreign key tables of the child table

These tables require full integrity processing to bring them out of the set integrity pending state. If you roll forward both table spaces simultaneously, the constraint remains 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 inconsistency can happen in the following cases:

- 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 that is 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 prevents 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 were removed. You can no longer roll forward the table space 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.)

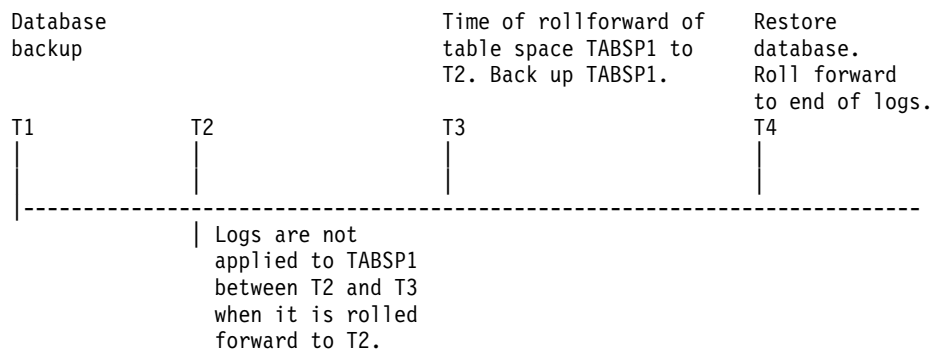


Figure 30. Table space backup requirement

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 timestamp 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 that was 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 being 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 forward that table space 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 forward TABSP1 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 forward the database.

Because you restore the table space before you roll forward the database, resources are 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 do one of the following actions:

- Roll forward the table space 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 by restoring the database backup that was taken at time T1, and then roll forward the table space to a time that precedes time T3.
- Drop the table space.

In a partitioned database environment:

- You must simultaneously roll forward all parts of a table space 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 forward the database partitions, and then roll forward the table spaces.
- If you intend to roll forward a table space to the end of the logs, you do not have to restore it at each database partition; you must restore it at the database partitions that require recovery. If you intend to roll forward a table space to a point in time, however, you must restore it at each database partition.

In a database with partitioned tables:

- If you are rolling a table space that contains any piece of a partitioned table forward to a point in time, you must also roll forward all of the other table spaces in which that table resides to the same point in time. However, rolling forward a single table space containing a piece of a partitioned table to the end of logs is allowed. If a partitioned table has any attached, detached, or dropped data partitions, then a point-in-time rollforward operation must also include all table spaces for these data partitions. In order to determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSCAT.DATAPARTITIONS catalog view.

## Database rollforward operations in a Db2 pureScale environment

In a Db2 pureScale environment, each member has its own log stream; however, log streams from all members are required for successful execution of the **ROLLFORWARD DATABASE** command.

During a database rollforward operation, log records from all of the log streams are merged and replayed to make the database consistent. The point in time that you specify on the **ROLLFORWARD DATABASE** command is relative to the merged log stream. To restore the database to a consistent state, the specified time must be later than the *minimum recovery time* (MRT). The MRT is the earliest time during a rollforward operation when objects that are listed in the database catalog match the objects that physically exist on disk. For example, if you are restoring from an image that was created during an online backup operation, the specified point in time for the rollforward operation must be later than the time at which the online backup operation completed. This will ensure database consistency.

The specified point in time for the subsequent database rollforward operation must be greater than or equal to the MRT in the merged log stream; otherwise, the rollforward operation fails (SQL1276N), and the timestamp of the MRT is returned with the error message. Alternatively, you can use the END OF BACKUP option to automatically roll forward to the MRT.

It is recommended that the member clocks be synchronized; however, it might not be possible to synchronize them at all times. This can result in log records having the same time stamp, and merged log streams with log records that appear to be out of time stamp order. In a Db2 pureScale environment, a point-in-time database rollforward operation stops when it encounters the first log record whose time stamp is greater than the specified time stamp from any log stream, and it has processed the log record that corresponds to the MRT for the database.

An incomplete or interrupted rollforward operation leaves the database in rollforward pending state. In this case, issue another **ROLLFORWARD DATABASE** command. In a Db2 pureScale environment, subsequent **ROLLFORWARD DATABASE** commands can be run on the same or on a different member.

In a Db2 pureScale environment, if you want to perform a database restore operation into a new database using an online database backup image, the correct approach depends on whether all of the log files are available, or only log files from the backup image are available.

- If pre-existing log files or archived log files can be accessed, the following rollforward operation is appropriate:

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

**Note:** Before taking a backup, you need to ensure that the log archiving path is set to a shared directory so that all the members are able to access the logs for subsequent rollforward operations. If the archive path is not accessible from the member on which the rollforward is being executed, SQL1273N is returned. The following command is an example of how to set the log path to the shared directory:

```
db2 update db cfg using logarchmeth1  
DISK:/db2fs/gpfs1/svtdbm5/svtdbm5/ArchiveLOGS
```

(where *gpfs1* is the shared directory for the members and *ArchiveLOGS* is the actual directory that archives the logs.

- If the only log files that can be accessed come from the backup image, the following rollforward operation is appropriate:

```
db2 rollforward db dbname to end of backup and stop
```

This command replays all required log records to achieve the consistent database state that was in effect when the backup operation ended. You can also use this command if pre-existing log files or archived log files can be accessed, but it will stop at the point at which the backup operation ended; it will not use any extra logs that were generated after the backup operation ended.

A **ROLLFORWARD DATABASE** command specifying the END OF LOGS option in this case would return SQL1273N. A subsequent **ROLLFORWARD DATABASE** command with the STOP option is successful, and the database will be available, if the missing log files are not needed. However, if the missing log files are needed (and it is not safe to stop), the rollforward operation will again return SQL1273N.



## Example

Suppose that there are two members, M1 and M2. M2's clock is ahead of M1's clock by five seconds. M2's log stream contains the following log records:

A1 at 2010-04-03-14.21.56  
A2 at 2010-04-03-14.21.56  
B at 2010-04-03-14.21.58  
C at 2010-04-03-14.22.01

M1's log stream contains the following log records:

D at 2010-04-03-14.21.55  
E at 2010-04-03-14.21.56  
F at 2010-04-03-14.21.57

The minimum recovery time (MRT) for the database on M2 is at time 2010-04-03-14.21.55. Because M1's clock is five seconds slow, log records D, E, and F appear later in the merged log stream:

```
MRT: 2010-04-03-14.21.55 (M2)
A1:  2010-04-03-14.21.56 (M2)
A2:  2010-04-03-14.21.56 (M2)
B:    2010-04-03-14.21.58 (M2)
D:    2010-04-03-14.21.55 (M1) --> corresponding time on M2 is 14.22.00
C:    2010-04-03-14.22.01 (M2)
E:    2010-04-03-14.21.56 (M1) --> corresponding time on M2 is 14.22.01
F:    2010-04-03-14.21.57 (M1) --> corresponding time on M2 is 14.22.02
```

The alphabetic characters (A1, A2, B, and so on) represent the order in which the corresponding log records were actually written at run time (across members). Note that log records A1 and A2 from member M2 have the same time stamp; this can happen when the Db2 data server tries to optimize performance by including the commit log record from multiple transactions when data is written from the log buffer to a log file.

The following command returns SQL1276N (Database "test" cannot be brought out of rollforward pending state until rollforward has passed a point in time greater than or equal to "2010-04-03-14.21.55"):

```
db2 rollforward db test to 2010-04-03-14.21.54
```

But the following command rolls forward the database up to and including log record A2:

```
db2 rollforward db test to 2010-04-03-14.21.56
```

Because log records A1 and A2 both have a time stamp that is less than or equal to the time that was specified in the command, both are replayed. Log record B, whose time stamp (2010-04-03-14.21.58) is greater than the specified value (2010-04-03-14.21.56), stops the rollforward operation and is not replayed. Log record D is not replayed either, even though its time stamp is less than the specified value, because log record B's higher value (2010-04-03-14.21.58) was encountered first. The following command rolls forward the database up to and including log record D:

```
db2 rollforward db test to 2010-04-03-14.21.58
```

Log record C, whose time stamp (2010-04-03-14.22.01) is greater than the specified value (2010-04-03-14.21.58), stops the rollforward operation and is not replayed. Log record E is not replayed either, even though its time stamp is less than the specified value.

## Rolling forward through an add member operation

Starting in Version 10.5, a IBM Db2 pureScale Feature instance backup image that was taken before adding a member online can be restored and rolled forward without requiring a full offline database backup.

### Restrictions and requirements

- The **ROLLFORWARD** command must be initiated from a member that is currently in the database topology.
- The add member event must correspond to a member that is currently in the member topology.
- The table space **ROLLFORWARD** command can process one or more add member events in the transaction logs.
- If there is an online table space roll forward operation in progress for the database, you are not able to issue the first connection to a database on a newly added member.

### Example

For examples of roll forward operations, see “Restore and roll forward through a topology change” on page 376

## Monitoring a rollforward operation

You can use the **db2pd** or the **LIST UTILITIES** command to monitor the progress of rollforward operations on a database.

### Procedure

- Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter  
`LIST UTILITIES SHOW DETAIL`
- Issue the **db2pd** command and specify the **-recovery** parameter:  
`db2pd -recovery`

### Results

For rollforward recovery, there are two phases of progress monitoring: FORWARD and BACKWARD. During the FORWARD phase, log files are read and the log records are applied to the database. For rollforward recovery, when this phase begins UNKNOWN is specified for the total work estimate. The amount of work processed in bytes is updated as the process continues.

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

### Example

The following is an example of the output for monitoring the performance of a rollforward operation using the **db2pd** command:

```
Recovery:
Recovery Status      0x00000401
Current Log          S0000005.LOG
Current LSN          0000001F07BC
Current LSO          000002551BEA
Job Type             ROLLFORWARD RECOVERY
Job ID               7
```

```

Job Start Time      (1107380474) Wed Feb  2 16:41:14 2005
Job Description     Database Rollforward Recovery
Invoker Type       User
Total Phases       2
Current Phase      1

```

```

Progress:
Address           PhaseNum Description StartTime           CompletedWork TotalWork
0x00000000200667160 1      Forward   Wed Feb  2 16:41:14 2005 2268098 bytes Unknown
0x00000000200667258 2      Backward   NotStarted              0 bytes      Unknown

```

The following is an example of the output for monitoring the performance of a database rollforward operation using the **LIST UTILITIES** command with the SHOW DETAIL option:

```

ID                      = 7
Type                    = ROLLFORWARD RECOVERY
Database Name           = TESTDB
Member Number           = 0
Description              = Database Rollforward Recovery
Start Time              = 01/11/2012 16:56:53.770404
State                   = Executing
Invocation Type         = User
Progress Monitoring:
  Estimated Percentage Complete = 50
  Phase Number                 = 1
    Description                 = Forward
    Total Work                  = 928236 bytes
    Completed Work              = 928236 bytes
    Start Time                  = 01/11/2012 16:56:53.770492

  Phase Number [Current]      = 2
    Description                = Backward
    Total Work                 = 928236 bytes
    Completed Work             = 0 bytes
    Start Time                 = 01/11/2012 16:56:56.886036

```

The following is an example of the output for monitoring the performance of a table space rollforward operation using the **LIST UTILITIES** command with the SHOW DETAIL option:

```

ID                      = 17
Type                    = ROLLFORWARD RECOVERY
Database Name           = TESTDB
Member Number           = 0
Description              = Offline Tablespace Rollforward Recovery: 3
Start Time              = 01/11/2012 17:04:27.269171
State                   = Executing
Invocation Type         = User
Progress Monitoring:
  Estimated Percentage Complete = 63
  Phase Number                 = 1
    Description                 = Forward
    Total Work                  = 142
    Completed Work              = 90
    Start Time                  = 01/11/2012 17:04:27.269283

  Phase Number [Current]      = 2
    Description                = Backward
    Total Work                 = 0
    Completed Work             = 0
    Start Time                 = Not Started

```

## Authorization required for rollforward

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

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.

## Rollforward sessions - CLP examples

You can issue rollforward commands from the Command Line Prompt. Before issuing a rollforward command, you might find it helpful to review some sample sessions.

### Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands are:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

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

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected before you stop it, so that you do not miss any logs.

If the rollforward command encounters an error, the rollforward operation will not complete. The error will be returned, and you will then be able to fix the error and reissue the command. If, however, you are unable to fix the error, you can force the rollforward to complete by issuing the following:

```
db2 rollforward db sample complete
```

This command brings the database online at the point in the logs before the failure.

### Example 2

Roll the database forward to the end of the logs (two table spaces have been restored):

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

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

### Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online
db2 rollforward db sample to 1998-04-03-14.21.56 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 and stop
overflow log path (/logs)
```

### Example 5

In the following example, there is a database called sample. The database is backed up and the recovery logs are included in the backup image; the database is restored; and the database is rolled forward to the end of backup timestamp.

Back up the database, including the recovery logs in the backup image:

```
db2 backup db sample online include logs
```

Restore the database using that backup image:

```
db2 restore db sample
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup
```

### Example 6 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

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

This returns warning SQL1271 ("Database is recovered but one or more table spaces are offline on database partitions 0 and 2.").

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

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

### Example 7 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with a single system view backup; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Perform a single system view (SSV) backup:

```
db2 backup db sample on all nodes online include logs
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample taken at 1998-04-03-14.21.56"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

### Example 8 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with one command using db2\_all; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Back up all the database partitions with one command using db2\_all:

```
db2_all "db2 backup db sample include logs to //dir/"
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample from //dir/"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

### Example 9 (partitioned database environments)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

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

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on  
dbpartitionnums (0, 2) tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop  
tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together.

**Note:** With table space rollforward to a point in time, the dbpartitionnum clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop  
tablespace(TBS1)
```

This completes successfully.

### Example 10 (partitioned database environments)

After restoring a table space on all database partitions, roll forward to PIT2, but do not specify AND STOP. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all dbpartitionnums **

db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

### Example 11 (partitioned database environments)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the db2nodes.cfg file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the db2nodes.cfg file.

### Example 12 (partitioned database environments)

Rollforward recover six small table spaces that reside on a single database partition database partition group (on database partition 6):

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

### Example 13 (Partitioned tables - Rollforward to end of log on all data partitions)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. All table spaces can be rolled forward to END OF LOGS.

```
db2 rollforward db PBARDB to END OF LOGS and stop
tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

### Example 14 (Partitioned tables - Rollforward to end of logs on one table space)

A partitioned table is created initially using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. Table space tbsp4 becomes corrupt and requires a restore and rollforward to end of logs.

```
db2 rollforward db PBARDB to END OF LOGS and stop tablespace(tbsp4)
```

This completes successfully.

### Example 15 (Partitioned tables - Rollforward to PIT of all data partitions including those added, attached, detached or with indexes)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, attaches data partitions from the table in tbsp5, and detaches data partitions from tbsp1. The user performs a rollforward to PIT with all the table spaces used by the partitioned table including those table spaces specified in the INDEX IN clause.

```
db2 rollforward db PBARDB to 2005-08-05-05.58.53 and stop
tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

### Example 16 (Partitioned tables - Rollforward to PIT on a subset of the table spaces)

A partitioned table is created using three table spaces (tbsp1, tbsp2, tbsp3). Later, the user detaches all data partitions from tbsp3. The rollforward to PIT is only permitted on tbsp1 and tbsp2.

```
db2 rollforward db PBARDB to 2005-08-05-06.02.42 and stop
tablespace( tbsp1, tbsp2)
```

This completes successfully.

---

## Data recovery with IBM Tivoli Storage Manager (TSM)

When calling the **BACKUP DATABASE** or **RESTORE DATABASE** commands, you can specify that you want to use the IBM Tivoli Storage Manager (TSM) product to manage database or table space backup or restore operation.

The minimum required level of TSM client API is Version 4.2.0, except on the following:

- 64-bit Solaris systems, which require TSM client API Version 4.2.1.
- 64-bit Windows operating systems, which require TSM client API Version 5.1.
- All Windows x64 systems, which require TSM client API Version 5.3.2.
- 32-bit Linux for IBM Power Systems, which requires TSM client API Version 5.1.5 or later.
- 64-bit Linux for IBM Power Systems, which requires TSM client API Version 5.2.2 or later.
- 64-bit Linux on AMD Opteron systems, which require TSM client API Version 5.2.0 or later.
- Linux for zSeries, which requires TSM client API Version 5.2.2 or later.

**Important:** Even though these Tivoli Storage Manager release levels work with Db2, they might not be supported by Tivoli Storage Manager

## Configuring a Tivoli Storage Manager client

Before the Db2 database manager can use an IBM Tivoli Storage Manager (TSM) client to manage database or table space backup or restore operations, you must configure the TSM environment.



## Before you begin

A functioning TSM client and server must be installed and configured. In addition, the TSM client API must be installed on each Db2 database server. TSM client proxy nodes are supported if the TSM server has been configured to support them. For information on server configuration and proxy node support, see “Considerations for using Tivoli Storage Manager” on page 439 or refer to the Tivoli documentation.

**Note:** Tivoli Storage Manager (TSM) Version 7.1.8 and Version 8.1.2 introduce significant enhancements for improved security between client and server communication. When the TSM server is upgraded to Version 7.1.8 (or higher versions) or 8.1.2 (or higher versions) and configured with the improved security protocol, and the TSM backup-archive client is upgraded to Version 7.1.8 (or higher versions) or Version 8.1.2 (or higher versions), the security settings for the backup-archive client must be re-configured to work with the new security enhancements on the server. Failure to re-configure the client may result in a TSM authentication error code 927 or other errors. For details, refer to the New Features and Updates of TSM Version 7.1.8: [https://www.ibm.com/support/knowledgecenter/SSGSG7\\_7.1.8/client/r\\_new\\_for\\_version.html](https://www.ibm.com/support/knowledgecenter/SSGSG7_7.1.8/client/r_new_for_version.html), or TSM Version 8.1.2: [https://www.ibm.com/support/knowledgecenter/SSEQVQ\\_8.1.2/client/r\\_new\\_for\\_version.html](https://www.ibm.com/support/knowledgecenter/SSEQVQ_8.1.2/client/r_new_for_version.html).

## Procedure

To configure the TSM environment for use by Db2 database systems:

1. Set the environment variables used by the TSM client API:

### **DSMI\_DIR**

Identifies the user-defined directory path where the API trusted agent file (dsmtca) is located.

### **DSMI\_CONFIG**

Identifies the user-defined directory path to the dsm.opt file, which contains the TSM user options. Unlike the other two variables, this variable should contain a fully qualified path and file name.

### **DSMI\_LOG**

Identifies the user-defined directory path where the error log (dsierror.log) will be created.

**Note:** In a multi-partition database environment, these settings must be specified in the sqllib/userprofile file.

2. If any changes are made to these environment variables and the database manager is running, stop and restart the database manager. For example:
  - Stop the database manager using the **db2stop** command.
  - Start the database manager using the **db2start** command.
3. Depending on the server's configuration, a Tivoli client might require a password to interface with a TSM server.

If the TSM environment is configured to use PASSWORDACCESS=generate, the Tivoli client needs to have its password established.

The executable file dsmapiw is installed in the sqllib/adsm directory of the instance owner. This executable allows you to establish and reset the TSM password.

To execute the **dsmapipw** command, you must be logged in as the local administrator or “root” user. When this command is executed, you will be prompted for the following information:

- *Old password*, which is the current password for the TSM node, as recognized by the TSM server. The first time you execute this command, this password will be the one provided by the TSM administrator at the time your node was registered on the TSM server.
- *New password*, which is the new password for the TSM node, stored at the TSM server. (You will be prompted twice for the new password, to check for input errors.)

**Note:** Users who invoke the **BACKUP DATABASE** or **RESTORE DATABASE** commands do not need to know this password. You only need to run the **dsmapipw** command to establish a password for the initial connection, and after the password has been reset on the TSM server.

## What to do next

Depending on your backup and log archiving strategies, you might need to perform additional steps to configure the TSM clients if you want to use proxy nodes. Proxy nodes enable you to consolidate backups and log archives of databases existing on multiple client nodes or under multiple users to a common target nodename on the TSM server. This configuration is useful when the administrator or computer that performs the backup can change over time, such as with clusters. The *asnodename* option also allows data to be restored from a different computer or from a user different than the one that performed the backup.

If you want to use TSM in your Db2 pureScale environment, proxy node configurations are recommended because each member can be represented as a TSM client or node and be mapped to a common proxy node.

If you do not want to use proxy nodes by default, no additional client setup is required. When you want to perform backup or restore operations using proxy nodes, specify the *asnodename* value in the **OPTIONS** parameter when invoking the **BACKUP DATABASE** or **RESTORE DATABASE** commands.

If you want to use TSM proxy nodes by default, use the following methods:

- Update database configuration parameters to use different proxy nodes for different databases.
- Update the *dsm.sys* file to use the same proxy node for all the users and databases on a machine.

**Note:** Every user-host combination using the same TSM proxy name will appear as the same Db2 instance to TSM. This can mean that if multiple Db2 instances use the same database name in a TSM client-node proxy configuration, then they can potentially overwrite each other's log archives and backup images. To avoid this:

- Create a different proxy hostname for each Db2 instance.
- Do not use TSM's client-node proxy feature if multiple Db2 instances might create databases using the same TSM proxy name.

**TSM client setup using *vendoropt*, *logarchopt1*, and *logarchopt2***

You can set one or more of the following database configuration parameters to enable different proxy node settings for each database:

- To enable commands using TSM (such as backup and restore) to use proxy nodes, specify the **asnodename** option in the **vendoropt** database configuration parameter, as follows:

```
db2 update db cfg for dbname using vendoropt "'-asnodename=proxynode'"
```

where *proxynode* is the name of the shared TSM proxy node.

- To configure log archiving to the TSM server, set the **logarchmeth1** database configuration parameter to TSM and specify the name of the proxy node as the **asnodename** value in the **logarchopt1** database configuration parameter, as follows:

```
db2 update db cfg for dbname using logarchmeth1 tsm
logarchopt1 "'-asnodename=proxynode'"
```

where *proxynode* is the name of the shared TSM proxy node.

You can make similar updates to the **logarchmeth2** and **logarchopt2** database configuration parameters.

In Db2 pureScale environments, these database configuration parameters are global parameters and you can set them from any member.

#### TSM client setup method using the **dsm.sys** file

1. Edit the **dsm.sys** file and add the proxy node information, as follows:

```
asnodename proxynode
```

where *proxynode* is the name of the shared TSM proxy node.

2. Ensure that the **dsm.opt** file specified in the **DSMI\_CONFIG** path contains the name of the TSM server, as follows:

```
servername servername
```

where *servername* is the TSM server name.

## Considerations for using Tivoli Storage Manager

You can use Tivoli Storage Manager to create backup images of a Db2 database. When you are deciding which backup tool to use for your database, you must consider the features and restrictions of each available option.

- To use specific features within Tivoli Storage Manager (TSM), you might be required to give the fully qualified path name of the object using the feature. (Remember that on Windows operating systems, the \ will be used instead of /.) The fully qualified path name of:
  - A full database recovery object is: */database/DBPARTnnn/FULL\_BACKUP.timestamp.seq\_no*
  - An incremental database recovery object is: */database/DBPARTnnn/DB\_INCR\_BACKUP.timestamp.seq\_no*
  - An incremental delta database recovery object is: */database/DBPARTnnn/DB\_DELTA\_BACKUP.timestamp.seq\_no*
  - A full table space recovery object is: */database/DBPARTnnn/TSP\_BACKUP.timestamp.seq\_no*
  - An incremental table space recovery object is: */database/DBPARTnnn/TSP\_INCR\_BACKUP.timestamp.seq\_no*
  - An incremental delta table space recovery object is: */database/DBPARTnnn/TSP\_DELTA\_BACKUP.timestamp.seq\_no*

where *database* is the database alias name, and *DBPARTnnn* is the database partition 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.
- If you perform an online backup operation and specify the **USE TSM** option and the **INCLUDE LOGS** option, a deadlock can occur if the two processes try to write to the same tape drive at the same time. If you are using a tape drive as a storage device for logs and backup images, you need to define two separate tape pools for TSM, one for the backup image and one for the archived logs.
- To use client proxy nodes, the TSM administrator must complete the following steps on the TSM server:
  1. If the Db2 clients are not already registered with the TSM server, register each client using the **register node** TSM command.
  2. Register a (virtual) common TSM nodename to be used by the group of clients with the TSM server using the **register node** TSM command.
  3. Grant proxy authority to all computers in the group using the **grant proxynode** TSM command.

For information on how to set up proxy node clients, see “Configuring a Tivoli Storage Manager client” on page 436 or refer to the Tivoli documentation.

- When taking incremental backups where only a small number of pages have been modified, it might be necessary to increase the TSM parameter **IDLETIMEOUT** to be larger than the time it takes to complete the backup of the largest tablespace. This prevents TSM from closing the session prior to the completion of the incremental backup.

---

## Db2 Advanced Copy Services (ACS)

Db2 Advanced Copy Services (ACS) enables you to use the fast copying technology of a storage device to perform the data copying part of backup and restore operations.

In a traditional backup or restore operation, the database manager copies data to or from disk or a storage device using operating system calls. Being able to use the storage device to perform the data copying makes the backup and restore operations much faster. A backup operation that uses Db2 ACS is called a snapshot backup.

To perform snapshot backup and restore operations, you need one of two things:

- A Db2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.
- For storage devices that are not supported, implement a custom script that allows your storage device to perform snapshot operations.

Db2 ACS interfaces with a storage solution, such as Tivoli Storage FlashCopy Manager, or IBM Spectrum Protect. For detailed instructions on the setup and usage of Tivoli Storage FlashCopy Manager, refer to the Tivoli documentation at: <https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Tivoli%20Storage%20FlashCopy%20Manager/page/Related%20Resources>

## Db2 Advanced Copy Services (ACS) best practices

Consider the following best practices when installing and configuring Db2 Advanced Copy Services (ACS).

### Specify a dedicated volume group for log paths

It is recommended that the log paths be contained within their own snapshot volume independent from the database directory and database containers.

### Specify one volume group for each database partition

In a partitioned database environment, each database partition must reside on a set of snapshot volumes independent of the other database partitions.

## Restrictions for FlashCopy Limited Function for xLinux and AIX SDK 1.0

If you are using Db2 Advanced Copy Services (ACS) with the FlashCopy Limited Function for xLinux and AIX SDK 1.0 that ships with Db2, you should be aware of some restrictions.

Volume sharing is not supported. If a database partition resides on the same storage volume as any other database partition, snapshot operations are not permitted. In addition, in order to use FlashCopy Limited Function for xLinux and AIX SDK 1.0 database and log volumes must reside on a file system which supports freeze and thaw requests. Table 21 lists a number of functional restrictions that can be avoided by obtaining a full license for IBM Tivoli Storage Manager (TSM) or IBM Spectrum Protect.

*Table 21. Comparison of supported features with the FlashCopy Limited Function for xLinux and AIX SDK 1.0 that ships with Db2 with the full version of the IBM Tivoli Storage Manager (TSM) product*

| Functional item                                                  | FlashCopy Limited Function for xLinux and AIX SDK 1.0 support                                 | Tivoli Storage FlashCopy Manager support                                                                                                             |
|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Local snapshot backup versions                                   | Maximum of two snapshot versions supported.                                                   | No product limit. Tivoli Storage FlashCopy Manager supports as many versions as the storage device and available resources allow.                    |
| Snapshot support integrated with mirroring                       | No support.                                                                                   | Support for snapshots from either source or target mirror sets for AIX Logical Volume Manager (LVM) mirroring.                                       |
| Integrated backup of snapshot image to tape                      | No integrated support. Traditional and snapshot backups are complementary but not integrated. | Fully integrated support for backup of snapshot image to TSM. A single backup command can drive the snapshot backup together with the backup to TSM. |
| Backup to tape offloaded from production server                  | No integrated support for backup to tape.                                                     | Fully integrated support for performing backup to TSM from a secondary host.                                                                         |
| Integrated backup of Tivoli Storage FlashCopy Manager repository | No support. External backup can be done when repository is inactive or shut down.             | Automatic backup to TSM.                                                                                                                             |

For more information about Tivoli Storage FlashCopy Manager or Spectrum Protect, consult the most recent documentation at [this developerWorks link](#).

## Enabling Db2 Advanced Copy Services (ACS)

To use Db2 Advanced Copy Services (ACS), or perform snapshot backup operations, you must install, activate, and configure a Db2 ACS API Driver for your storage device.

### Before you begin

To perform snapshot backup and restore operations, you need one of two things:

- A Db2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.
- For storage devices that are not supported, implement a custom script that allows your storage device to perform snapshot operations.

### Procedure

1. Install Db2 ACS API Driver. See: “Installing Db2 Advanced Copy Services (ACS) API Driver.”
2. Create the database manager instance or instances with which you will use Db2 ACS.

When you create a new database manager instance, a directory called `acs` is created in the new instance `sqllib` directory. Because each database manager instance has an `acs` directory, you can configure each database manager instance differently.

3. For each database manager instance with which you will use Db2 ACS, perform the following steps:
  - a. Activate Db2 ACS. See: “Activating Db2 Advanced Copy Services (ACS) Driver manually” on page 443.
  - b. Configure Db2 ACS. See: “Configuring Db2 Advanced Copy Services (ACS) Driver” on page 444.

### Results

After you have enabled Db2 ACS, you have to configure your storage solution before you can perform snapshot backups. For instructions on configuring and using Tivoli Storage FlashCopy Manager or FlashCopy Limited Function for xLinux and AIX SDK 1.0 with your Db2 product, consult the most recent documentation at: Tivoli Documentation Central.

## Installing Db2 Advanced Copy Services (ACS) API Driver

The files and libraries that are required for Db2 Advanced Copy Services (ACS) API Driver must be installed manually.

### Before you begin

Before installing ACS, you must have the following libraries installed:

On AIX:

- `ln -s /opt/freeware/lib/powerpc-ibm-aix5.3.0/libgcc_s.a /usr/lib/libgcc_s.a`

You should also review the following topics:

- “Db2 ACS installation and configuration best practices”. See “Db2 Advanced Copy Services (ACS) best practices” on page 441
- “Restrictions for embedded version of Tivoli Storage FlashCopy Manager”. See “Restrictions for FlashCopy Limited Function for xLinux and AIX SDK 1.0” on page 441

On Red Hat Enterprise Linux:

- `ln -s libssl.so.0.9.8xxx libssl.so.0.9.8`
- `ln -s libcrypto.so.0.9.8xxx libcrypto.so.0.9.8`
- `ln -s libssl.so.0.9.8xxx libssl.so`
- `ln -s libssl.so.0.9.8xxx libssl.so.0`

### Restrictions

Db2 ACS API Driver supports a subset of hardware and operating systems that IBM Data Server supports. For a list of hardware and operating systems that Db2 ACS supports, consult the relevant documentation at this [developerWorks link](#).

### Procedure

1. Download from PPA
2. Unpack
3. Install

### What to do next

After Db2 ACS has been installed, you must activate Db2 ACS and configure Db2 ACS. See “Activating Db2 Advanced Copy Services (ACS) Driver manually” and “Configuring Db2 Advanced Copy Services (ACS) Driver” on page 444.

### Activating Db2 Advanced Copy Services (ACS) Driver manually

Before you can use Db2 Advanced Copy Services (ACS) to perform a snapshot backup for a given database manager instance, Db2 ACS functionality must be activated on that instance.

The database manager automatically calls **setup\_db2.sh** to activate Db2 ACS functionality during database manager instance creation.

### Before you begin

Before you can activate Db2 ACS, it must already be installed and the database manager instance or instances with which you will use Db2 ACS must have been created.

### About this task

The database manager automatically calls **setup\_db2.sh** to activate Db2 ACS functionality during database manager instance creation. You must explicitly issue the setup script to install DB2® Limited Function for xLinux and AIX SDK 1.0.

You can also activate Db2 ACS manually.

## Procedure

To activate Db2 ACS manually, run the **setup\_db2.sh** script as a user with root authority, and with appropriate parameters to activate Db2 ACS. For more information about **setup\_db2.sh**, see: “setup\_db2.sh script” on page 445.

## Results

One important result of running the **setup\_db2.sh** script is that the ownership and permissions of Db2 ACS executable files in the `sql1ib/acs` directory are verified.

## What to do next

After you have activated Db2 ACS, you must configure Db2 ACS before you can perform snapshot backup operations.

## Configuring Db2 Advanced Copy Services (ACS) Driver

Before you can use Db2 Advanced Copy Services (ACS) to perform a snapshot backup, you must configure Db2 ACS. You use configuration files to configure Db2 ACS.

## Before you begin

You must perform the following tasks before you can configure Db2 ACS:

1. Install Db2 ACS. Refer to “Installing Db2 Advanced Copy Services (ACS) API Driver” on page 442.
2. Create the database manager instance or instances with which you will use Db2 ACS..
3. Activate Db2 ACS. Refer to “Activating Db2 Advanced Copy Services (ACS) Driver manually” on page 443.

## Procedure

Run the **setup\_db2.sh** script from the `sql1ib/acs` directory without any parameters. This will lead you through an interactive, text-based wizard that will configure Db2 ACS. The wizard creates a configuration profile file and modifies the `/etc/initab` on the machine to trigger the launch of the Db2 ACS daemons.

## Configuring the Db2 Advanced Copy Services (ACS) directory:

When you create a new database manager instance, a directory called `acs` is created in the new instance `sql1ib` directory. Db2 Advanced Copy Services (ACS) uses this `acs` directory to store configuration files like the target volume control file and the shared repository for recovery objects.

There are restrictions on the ways you can alter or configure this `acs` directory.

## About this task

1. The `acs` directory must not be involved in any Db2 ACS or snapshot backup operation.
2. The `acs` directory can be NFS-exported and NFS-shared on all database partitions and on the backup system for a snapshot backup using IBM Tivoli Storage Manager (TSM) or Spectrum Protect.



## setup\_db2.sh script

The **setup\_db2.sh** script activates and configures Db2 Advanced Copy Services (ACS) Driver, and manually installs Flashcopy Limited Function for xLinux and AIX SDK 1.0

### Location

The script **setup\_db2.sh** is located in the `sql11ib/acs` directory.

### Syntax

Here is the syntax for **setup\_db2.sh**:

```
►►—setup_db2.sh—┐—————►
                  └—a—action—-d—Db2_Instance_Directory—┘
```

where *action* can be one of:

#### disable

This option stops Flashcopy Limited Function and removes all entries from `/etc/inittab`. To use this option, you must have root authority or be the instance owner.

#### install

This option installs Flashcopy Limited Function. To use this option, you must have root authority.

#### start

This option starts a previously installed and configured version of Flashcopy Limited Function. To use this option, you must have root authority or be the instance owner.

#### stop

This option stops the currently running version of Flashcopy Limited Function. To use this option, you must have root authority or be the instance owner.

### Usage

The database manager automatically calls **setup\_db2.sh** to activate Db2 ACS functionality during database manager instance creation. You must explicitly issue the setup script to install Flashcopy Limited Function for xLinux and AIX SDK 1.0.

You can call the **setup\_db2.sh** script manually to do the following tasks:

#### Activate Db2 ACS

You can activate Db2 ACS Driver by running the **setup\_db2.sh** script with the parameters described previously, as a user with root authority.

#### Configuring Db2 ACS

You can configure Db2 ACS Driver by running the **setup\_db2.sh** script without any parameters. If you run the **setup\_db2.sh** script without any parameters, then a wizard will lead you through Db2 ACS configuration.

#### Installing Flashcopy Limited Function for xLinux and AIX 1.0

On Linux and AIX systems, you need to manually run the **setup\_db2.sh** script to install Flashcopy Limited Function.

One important result of running the **setup\_db2.sh** script is that the ownership and permissions of Db2 ACS executable files in the `sqllib/acs` directory are verified.

## Manually installing Tivoli Storage FlashCopy Manager (Linux)

The version of Tivoli Storage FlashCopy Manager, which you are able to separately download with your IBM product, must be manually installed on Linux and AIX operating systems.

### Before you begin

To install Tivoli Storage FlashCopy Manager, you need to have root authority. In addition, the Db2 instance with which you are going to use Db2 ACS must already have been created.

### About this task

This task instructs you how to obtain the Linux-specific packages for Tivoli Storage FlashCopy Manager and how to issue the setup script, which copies the binary files to the instance-specific installation directory and sets the appropriate access rights for the binary files.

### Procedure

1. Locate the Linux packages for Tivoli Storage FlashCopy Manager on the separate CD labelled `fcm_linux`. Alternatively, you can download the `fcm_linux` packages from the same website where you obtain the product image.
2. Unzip the packages to the ACS directory in the install path, for example `/opt/IBM/db2/V10.1/acs`
3. Call the **setup\_db2.sh** setup script as follows:  

```
setup_db2.sh -a install -d Instance_directory
```

### Results

You should now be able to perform snapshot backups. You should be aware that the version of Tivoli Storage FlashCopy Manager that is embedded with your Db2 product has some limitations compared with the full version of the product that you get with the IBM Tivoli Storage Manager product.

## Db2 Advanced Copy Services (ACS) scripted interface

If you want to perform snapshot operations with a storage device that does not provide a vendor library to implement the Db2 ACS API, you have to create your own script.

A script allows the Db2 ACS API to directly communicate with the storage system and create a snapshot of volumes which contain data and log files for a database. Afterward, you can use a different script to perform the complementary action of restoring the snapshot image, or even deleting the image.

By creating your own script for performing snapshots, you can use unsupported storage boxes, or boxes that are available before a vendor library is available for interfacing with Db2 ACS. A vendor library provides the necessary extensions for implementing snapshot-based backup and restore. A script serves a similar role. With the improved interfacing with scripts introduced in V10.5, Db2 removes the need for the script to account for some of the more error prone actions, like suspending and resuming operations when taking a snapshot backup. Like

snapshot operations with supported storage hardware, snapshot operations that use scripts generate a history file entry, meaning that you can monitor the success or failure of snapshot backups.

The Db2 ACS API is wrapped in the library for Db2 ACS, which is included with the Db2 product. The library for ACS writes the protocol files to the protocol file repository and invokes the script that you specify for your snapshot operation.

### Db2 Advanced Copy Services (ACS) protocol file

The Db2 Advanced Copy Services (ACS) protocol files are created by the library for Db2 ACS and contain information that is needed by scripts for snapshot operations.

The protocol files are located in the protocol file repository. You should create a directory for this repository before performing your snapshot operation. You specify the repository by using the **options** parameter with the relevant command. If you do not create a directory, the protocol file repository will be the same directory that the script is located.

The protocol files serve two purposes:

- They show the progress of the running operation. In the case of failed operations, they also contain some information that you can use for debugging.
- They contain information and options provided from the library for Db2 ACS to the script. Some of the information, such as the metadata string, is also needed by the library for Db2 ACS or Db2 to restore the snapshot.

A protocol file is divided into different sections, each of which shows the progress and options of each function call. The output in each section contains the following information:

- Function name. For example `db2ACSInitialize`  
For a list of functions see Db2 Advanced Copy Services (ACS) API functions
- Beginning and ending timestamp for the function call
- Commands that were used to invoke the script, in the following format  

```
cmd: path_to_script -a action
      -c protocol_file_repository/protocol_file_name.cfg
```
- Options that were given in the function calls. See Table 22 for a list and description of the options.

Table 22. Options written by the library for Db2 ACS

| Key name             | Description                                                                                                                                                                                                                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ACTION               | Action that is being performed: <ul style="list-style-type: none"> <li>• DB2ACS_ACTION_READ_BY_GROUP during restore of parts of the database, in particular restore excluding log files</li> <li>• DB2ACS_ACTION_READ_BY_OBJECT during restore of the whole database, DB2ACS_ACTION_WRITE during snapshot</li> </ul> |
| APP_OPTIONS          | Hex value that combines the DB2BACKUP_*                                                                                                                                                                                                                                                                              |
| DATAPATH_AUTOSTORAGE | Key for each storage path that is configured in the database                                                                                                                                                                                                                                                         |
| DATAPATH_DB          | Database paths configured in the database                                                                                                                                                                                                                                                                            |

Table 22. Options written by the library for Db2 ACS (continued)

| Key name                | Description                                                                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| DATAPATH_GENERIC        | <ul style="list-style-type: none"> <li>Additional paths configured in the database</li> <li>Placeholder for future types</li> </ul>                    |
| DATAPATH_LOCAL_DB       | Local database paths configured in the database                                                                                                        |
| DATAPATH_TBSP_CONTAINER | Key for each DMS container that is configured in the database                                                                                          |
| DATAPATH_TBSP_DEVICE    | Key for each raw device that is configured in the database                                                                                             |
| DATAPATH_TBSP_DIR       | Key for each SMS storage path that is configured in the database                                                                                       |
| DB_NAME                 | Name of the database for that the operation is done                                                                                                    |
| DBPARTNUM               | Database partition number to be operated on                                                                                                            |
| DB2BACKUP_MODE          | Whether the backup is offline or online                                                                                                                |
| DB2BACKUP_LOGS          | Whether log files are included in or excluded from the backup. If logs are excluded, LOG_DIR and MIRRORLOG_DIR are not contained in the protocol file. |
| DELETE_OBJ_ID           | Object ID of the object to be deleted                                                                                                                  |
| EXTERNAL_OPTIONS        | Lists any options that you specify in the backup and restore command and are automatically copied into to the custom script.                           |
| EXTERNAL_SCRIPT         | Name of the script for the snapshot operation                                                                                                          |
| FIRST_ACTIVE_LOG_CHAIN  | The log chain of the first active log                                                                                                                  |
| FIRST_ACTIVE_LOG_ID     | Number of first active log of the database during the time the snapshot was taken                                                                      |
| INSTANCE                | Name of the instance for the database                                                                                                                  |
| LOGPATH_MIRROR          | Mirror log directory                                                                                                                                   |
| LOGPATH_PRIMARY         | Log directory of the database                                                                                                                          |
| METADATA                | String that represents the Base64 encoded meta data memory block                                                                                       |
| METADATA_CHECKSUM       | Checksum of the Base64 metadata string                                                                                                                 |
| METADATA_SIZE           | Size of the encoded metadata string                                                                                                                    |
| METADATA_DECODED_SIZE   | Size of the decoded metadata block                                                                                                                     |
| OBJ_DB2ID_LEVEL         | Fix pack level that was used during the backup                                                                                                         |
| OBJ_DB2ID_RELEASE       | Release level of the Db2 product that was used during the backup                                                                                       |
| OBJ_DB2ID_VERSION       | Version of the Db2 product that was used during backup                                                                                                 |
| OBJ_HOST                | Host server where the database partition resides                                                                                                       |

Table 22. Options written by the library for Db2 ACS (continued)

| Key name          | Description                                                                                                                                                          |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OBJ_ID            | Unique identifier for each stored object                                                                                                                             |
| OBJ_OWNER         | Owner of the object                                                                                                                                                  |
| OBJ_TYPE          | Snapshot                                                                                                                                                             |
| OPERATION         | Operation identifier: <ul style="list-style-type: none"> <li>• Delete</li> <li>• Restore</li> <li>• Snapshot</li> </ul>                                              |
| QUERY_DB          | Name of the database that is queried for                                                                                                                             |
| QUERY_HOST        | Name in the host in the object                                                                                                                                       |
| QUERY_INSTANCE    | Name of the instance of the database that is contained in the backup image                                                                                           |
| QUERY_OWNER       | Owner of the object                                                                                                                                                  |
| QUERY_DBPARTNUM   | <ul style="list-style-type: none"> <li>• Number of the database partition backed up to the object</li> <li>• -1 for the generic case</li> </ul>                      |
| QUERY_TIMESTAMP   | Timestamp queried for                                                                                                                                                |
| QUERY_TYPE        | Type of the object to be queried, snapshot or hex code representing the type                                                                                         |
| RC_DELETE         | Return code of the deletion operation. A non-zero value indicates that an error happened in the section.                                                             |
| RC_OPERATION      | Return code of the complete backup operation. A non-zero value indicates that an error happened in the section.                                                      |
| RC_PREPARE        | Return code of the prepare action. A non-zero value indicates that an error happened in the section.                                                                 |
| RC_RESTORE        | Return code of the complete restore operation. A non-zero value indicates that an error happened in the section.                                                     |
| RC_SNAPSHOT       | Return code of the snapshot action. A non-zero value indicates that an error happened in the section.                                                                |
| RC_STORE_METADATA | Return code of store_metadata operation. A non-zero value indicates that an error happened in the section.                                                           |
| RC_VERIFY         | Return code of the verify action. A non-zero value indicates that an error happened in the section.                                                                  |
| RESULT_n_FILE     | The name of the <i>n</i> th file during query, delete and restore                                                                                                    |
| SIGNATURE         | Software level of the Db2 version being used                                                                                                                         |
| SYNC_MODE         | Writes NONE on single partition databases, PARALLEL if snapshot is taken in parallel on all nodes, SERIAL if the snapshot is taken on the nodes one after the other. |

## Example protocol file for a snapshot backup

This section contains an example protocol file written for a snapshot backup operation invoking the sample script. For illustrative purposes, it has been broken up into sections for each Db2 ACS API function that is a part of the operation.

### db2ACSInitialize

After loading the library for Db2 ACS and querying the version of the Db2 ACS API, the database manager establishes a Db2 ACS session by calling db2ACSInitialize(). This step is required for all operations.

The flags that are of most interest for monitoring purposes are:

- EXTERNAL\_SCRIPT: the name and path of the script
- DB\_NAME: the database name
- INSTANCE: the Db2 instance name
- DBPARTNUM: the database partition number

```
# =====
# db2ACSInitialize(): BEGIN [2012-11-30 08:15:45]
EXTERNAL_SCRIPT=/home/hotellnx99/jklauke/libacssc.sh
HANDLE=1354281345
START_TIME=1354281345
DB_NAME=SAMPLE
INSTANCE=jklaue
DBPARTNUM=0
SIGNATURE=SQL10020
EXTERNAL_OPTIONS=/home/hotellnx99/jklauke/repository 2ndoption
# db2ACSInitialize(): END
# =====
```

### db2ACSBeginOperation

The database manager calls db2ACSBeginOperation() to begin the specified operation (indicated in the OPERATION flag).

```
# =====
# db2ACSBeginOperation(): BEGIN [2012-11-30 08:15:45]
OPERATION=snapshot
# db2ACSBeginOperation(): END
# =====
```

### db2ACSPartition

The database manager calls db2ACSPartition(), which associates a group identifier with each of the paths listed by the database manager as belonging to a database partition. The library for Db2 ACS groups database path information for a single database partition together, so the partition ID is unique for every path. This makes it possible to take a snapshot at the file-set, file-system, and volume-group level. The path-related flags that are of interest in this section are:

- LOG\_DIR, MIRRORLOG\_DIR: the log paths
- DB\_PATH, LOCAL\_DB\_PATH: the database paths
- STORAGE\_PATH, CONT\_PATH, TBSP\_DIR

The SYSIBMADM.DBPATHS administrative view provides these same path types.

A number of flags provide information about the settings for the current operation:

- DB2BACKUP\_MODE: Offline or online backup
- DB2BACKUP\_LOGS: Exclude or include logs. In this example, the logs are included, so the sample customer script compresses the log files but in a different file than the other database files.

```
# =====
# db2ACSPartition(): BEGIN [2012-11-30 08:15:06]
OBJ_HOST=hotel1nx99
OBJ_OWNER=
OBJ_TYPE=snapshot
OBJ_DB2ID_LEVEL=0
OBJ_DB2ID_RELEASE=2
OBJ_DB2ID_VERSION=10
APP_OPTIONS=0x1000
DB2BACKUP_MODE=OFFLINE
DB2BACKUP_LOGS=INCLUDE
LOGPATH_PRIMARY=/home/hotel1nx99/jklauke/jklauke/NODE0000/SQL00001/LOGSTREAM0000/
DATAPATH_DB=/home/hotel1nx99/jklauke/jklauke/NODE0000/SQL00001/MEMBER0000/
DATAPATH_LOCAL_DB=/home/hotel1nx99/jklauke/jklauke/NODE0000/sqlbdir/
DATAPATH_DB=/home/hotel1nx99/jklauke/jklauke/NODE0000/SQL00001/
DATAPATH_AUTOSTORAGE=/home/hotel1nx99/jklauke/jklauke/NODE0000/SAMPLE/
# db2ACSPartition(): END
# =====
```

### db2ACSPprepare

The database manager calls db2ACSPprepare() to prepare to perform the snapshot. In the protocol file, the prepare section shows the command with which the script was invoked and the return code of the preparation.

```
# =====
# db2ACSPprepare(): BEGIN [2012-11-30 08:15:45]
# cmd: /home/hotel1nx99/jklauke/libacssc.sh -a prepare
#       -c /home/hotel1nx99/jklauke/repository/db2acs.SAMPLE.0.jklauke.1353420068.cfg
#       /home/hotel1nx99/jklauke/repository/2ndoption
RC_PREPARE=0
# db2ACSPprepare(): END
# =====
```

If this step completes successfully, the database manager puts the database in SET WRITE SUSPEND state (assuming the snapshot backup is online).

### db2ACSSnapshot

The database manager calls db2ACSSnapshot() to perform the snapshot. The protocol file shows the command used during the snapshot and the return code of the snapshot operation. This part of the protocol file shows the point at which the real snapshot is taken and the vendor tools are triggered that run the operations on the storage boxes.

Note that the content between ACTION=DB2ACS\_ACTION\_WRITE and RC\_SNAPSHOT is specific to the sample script, which compresses all of the paths shown in the db2ACSPartition section of the protocol file into one tar file and all log files (primary and mirror log files) to a second tar file.

```
# =====
# db2ACSSnapshot(): BEGIN [2013-01-15 10:18:23]
OBJ_ID=0
ACTION=DB2ACS_ACTION_WRITE
# cmd:/home/hotel1nx99/jklauke/sql/lib/samples/BARVendor/libacssc.sh -a snapshot
#       -c /home/hotel1nx99/jklauke/repository/db2acs.SAMPLE.0.jklauke.
#       1358263103.cfg
BACKUP_FILE=/home/hotel1nx99/jklauke/repository/SAMPLE.0.jklauke.
# 0.20130115101824.001.tar
# cmd: awk -F= '/^DATAPATH/
#       { print $2; }' /home/hotel1nx99/jklauke/repository/db2acs.SAMPLE.
#       0.jklauke.1358263103.cfg
#       | tar -Pcf /home/hotel1nx99/jklauke/repository/SAMPLE.0.jklauke.
#       0.20130115101824.001.tar
#       -T - 2>/dev/null && echo 0 || echo 1
# backup tar created, rc=0
# Logs to be included
BACKUP_LOGS=/home/hotel1nx99/jklauke/repository/SAMPLE.0.jklauke.
# 0.20130115101824.log.tar
# cmd: awk -F= '/^LOGPATH/ { print $2; }'
```

```

/home/hotel1nx99/jklauke/repository/db2acs.SAMPLE.0.jklauke.1358263103.cfg
| tar -Pcf /home/hotel1nx99/jklauke/repository/SAMPLE.0.jklauke.
0.20130115101824.log.tar
-T - 2>/dev/null && echo 0 || echo 1
# tar for logs created, rc=0
RC_SNAPSHOT=0
# db2ACSSnapshot(): END [2013-01-15 10:18:24]
# =====

```

After this step completes, the database manager puts the database in WRITE RESUME state.

### db2ACSVerify

The database manager calls db2ACSVerify() to verify that the snapshot backup succeeded. If your script contains a verify action, the library for Db2 ACS invokes your script. In the example script, the verify step only checks for the existence of the two tar files (if EXCLUDE LOGS were specified it would not check for the existence of the tar file for the logs).

```

# =====
# db2ACSVerify(): BEGIN [2012-11-30 08:15:08]
FIRST_ACTIVE_LOG_ID=2
FIRST_ACTIVE_LOG_CHAIN=3
# cmd: /home/hotel1nx99/jklauke/libacssc.sh -a verify
-c /home/hotel1nx99/jklauke/repository/db2acs_SAMPLE_1354281306_0.cfg
/home/hotel1nx99/jklauke/repository/2ndoption
# Backup '/home/hotel1nx99/jklauke/repository/SAMPLE.0.jklauke.
0.1354281306.001.tar' checked: looks okay
# Logs '/home/hotel1nx99/jklauke/repository/SAMPLE.0.jklauke.
0.1354281306.log.tar' checked: looks okay
RC_VERIFY=0
# db2ACSVerify(): END
# =====

```

If the script returns a non-zero return code, the following db2ACSSStoreMetaData() call is skipped and db2ACSEndOperation is called instead. In the case of the example script, the library for Db2 ACS invokes the script with the rollback action. For an example of this, see this section.

### db2ACSSStoreMetaData

The database manager calls db2ACSSStoreMetaData() to store metadata about the recovery object created by the operation. If your script contains a store\_metadata action, the library for Db2 ACS invokes your script to perform actions such as:

- backing up the protocol file (it has to exist for a snapshot to be restored, queried, or deleted)
- renaming the backup

```

# =====
# db2ACSSStoreMetaData(): BEGIN [2013-01-15 10:18:24]
START_TIME=1358263104
METADATA_SIZE=12024
METADATA=U1FMV...
METADATA_CHECKSUM=16941
# cmd: /home/hotel1nx99/jklauke/sql1lib/samples/BARVendor/libacssc.sh
-a store_metadata
-c /home/hotel1nx99/jklauke/repository/db2acs.SAMPLE.0.jklauke.1358263103.cfg
RC_STORE_METADATA=0
# db2ACSSStoreMetaData(): END [2013-01-15 10:18:24]
# =====

```

### db2ACSEndOperation

The database manager calls db2ACSEndOperation() to end the operation.



### Successful operations

The return code of 0 indicates that the snapshot operation was successful.

```
# =====
# db2ACSEndOperation(): BEGIN [2012-11-30 08:15:08]
RC_OPERATION=0
# db2ACSEndOperation(): END
```

### Failed operations

If the snapshot operation failed--that is, a call to the customer script had a non-zero return code or there was an internal error in the library for Db2 ACS--the db2ACSEndOperation section of the protocol file has a non-zero return code. If you specify a rollback action in your script, the script is called at this point. In the case of the sample script, the protocol file contains the following output:

```
# =====
# db2ACSEndOperation(): BEGIN [2013-01-18 05:26:06]
RC_OPERATION=1
# cmd:/home/hotellnx99/jklauke/sqllib/samples/BARVendor/libacssc.sh -a rollback
# -c /home/hotellnx99/jklauke/repository/db2acs.SAMPLE.0.jklauke.1358504766.cfg
# Delete old backup file :
# /home/hotellnx99/jklauke/repository/SAMPLE.0.jklauke.0.20130118052606.001.tar
# Delete old backup file :
# /home/hotellnx99/jklauke/repository/SAMPLE.0.jklauke.0.20130118052606.log.tar
RC_ROLLBACK=0
# db2ACSEndOperation(): END [2013-01-18 05:26:06]
# =====
```

### db2ACSTerminate

The database manager calls db2ACSTerminate() to terminate the session.

```
# =====
# db2ACSTerminate(): BEGIN [2012-11-30 08:15:08]
# db2ACSTerminate(): END
# =====
```

## Example protocol file for a snapshot restore

This section contains an example protocol file written for a snapshot restore operation invoking the sample script. A snapshot restore reads the protocol file for the snapshot backup, while at the same time writing new protocol files for the restore operation. If the restore is successful, those protocol files are deleted. For illustrative purposes, the following protocol file for a snapshot restore has been broken up into sections for each Db2 ACS API function that is a part of the operation.

### db2ACSInitialize

After loading the library for Db2 ACS and querying the version of the Db2 ACS API, the database manager establishes a Db2 ACS session by calling db2ACSInitialize(). This step is required for all operations.

The flags that are of most interest for monitoring purposes are:

- EXTERNAL\_SCRIPT: the name and path of the script
- DB\_NAME: the database name
- INSTANCE: the Db2 instance name
- DBPARTNUM: the database partition number

```
# =====
# db2ACSInitialize(): BEGIN [2012-11-30 08:27:38]
REPOSITORY_PATH=/home/hotellnx99/jklauke/repository/
EXTERNAL_SCRIPT=/home/hotellnx99/jklauke/libacssc.sh
HANDLE=1354282058
```

```

START_TIME=1354282058
DB_NAME=SAMPLE
INSTANCE=jklauke
DBPARTNUM=0
SIGNATURE=SQL10020
EXTERNAL_OPTIONS=/home/hotellnx99/jklauke/repository
# db2ACSInitialize(): END
# =====

```

### db2ACSBeginOperation

The database manager calls db2ACSBeginOperation() to begin the specified operation (indicated in the OPERATION flag).

```

# db2ACSBeginOperation(): BEGIN [2012-11-30 08:27:38]
OPERATION=restore
# db2ACSBeginOperation(): END
# =====

```

### db2ACSBeginQuery

The database manager calls db2ACSBeginQuery() to determine which snapshot backup objects are available to be used for the restore operation and to prepare the restore. The protocol file also shows the command with which the script was invoked for the prepare action and the return code of the preparation.

```

# =====
# db2ACSBeginQuery(): BEGIN [2012-11-30 08:27:38]
QUERY_TYPE=snapshot
QUERY_PARTNUM=0
QUERY_DB=SAMPLE
QUERY_INSTANCE=*
QUERY_HOST=*
QUERY_OWNER=*
QUERY_TIMESTAMP=20121130082717
# cmd: /home/hotellnx99/jklauke/libacssc.sh -a prepare
#       -c/home/hotellnx99/jklauke/db2acs.SAMPLE.0.jklauke.1353421208.cfg
#       /home/hotellnx99/jklauke/repository
RC_PREPARE=0
# db2ACSBeginQuery(): END
# =====

```

### db2ACSGetNextObject

The database manager calls db2ACSGetNextObject() to find an appropriate backup image for the given timestamp. The function is called iteratively and loops over the available files, giving information about each backup image to the database manager. The following output shows the looping over three protocol files:

```

# =====
# db2ACSGetNextObject(): BEGIN [2012-12-13 08:01:39]
RESULT_0_FILE=/home/hotellnx99/jklauke/repository/db2acs.SAMPLE.
0.jklauke.1355341475.cfg
# read result object with timestamp 20121212144436
# db2ACSGetNextObject(): END [2012-12-13 08:01:39]
# =====
# db2ACSGetNextObject(): BEGIN [2012-12-13 08:01:39]
RESULT_1_FILE=/home/hotellnx99/jklauke/repository/db2acs.SAMPLE.
0.jklauke.1355341690.cfg
# read result object with timestamp 20121212144811
# db2ACSGetNextObject(): END [2012-12-13 08:01:39]
# =====
# db2ACSGetNextObject(): BEGIN [2012-12-13 08:01:39]
RESULT_2_FILE=/home/hotellnx99/jklauke/repository/db2acs.SAMPLE.
0.jklauke.1355341892.cfg
# read result object with timestamp 20121212145133
# db2ACSGetNextObject(): END [2012-12-13 08:01:39]
# =====

```

### db2ACSRetrieveMetaData

The database manager calls db2ACSRetrieveMetaData() to retrieve all metadata about the backup image.

```
# =====
# db2ACSRetrieveMetaData(): BEGIN [2012-11-30 08:27:39]
GET_META_OBJ_ID=3
METADATA_DECODED_SIZE=9004
METADATA_CHECKSUM=14583
# db2ACSRetrieveMetaData(): END
# =====
```

### db2ACSSnapshot

The database manager calls db2ACSSnapshot() to perform the restore. The protocol file shows the commands and actions used by the script and the return code of the snapshot operation. The action can be one of two options:

- DB2ACS\_ACTION\_READ\_BY\_OBJECT. This indicates that the LOGTARGET INCLUDE FORCE options were specified with the **RESTORE DATABASE** command. The script uncompresses both tar files (one for the data and one for the logs). You also need to copy the disks used for the log files.
- DB2ACS\_ACTION\_READ\_BY\_GROUP. This indicates that the LOGTARGET EXCLUDE FORCE options were specified with the **RESTORE DATABASE** command. This shows the groups, or IDs of the file systems, for the groups that have to be restored. You must not copy the disks used for the log files.

```
# =====
# db2ACSSnapshot(): BEGIN [2012-11-30 08:27:40]
OBJ_ID=3
ACTION=DB2ACS_ACTION_READ_BY_OBJECT
# cmd:/home/hotellnx99/jklauke/libacssc.sh -a restore
#   -c /home/hotellnx99/jklauke/repository/db2acs_SAMPLE_1354282058_0.cfg
#   /home/hotellnx99/jklauke/repository
# cmd: tar -xf /home/hotellnx99/jklauke/repository/SAMPLE.0.jklauke.
# 0.20121130082717.001.tar
#   && echo 0 || echo 1
# tar extracted, rc=0
# cmd: tar -xf /home/hotellnx99/jklauke/repository/SAMPLE.0.jklauke.
# 0.20121130082717.log.tar
#   && echo 0 || echo 1
# logs extracted, rc=0
RC_RESTORE=0
# db2ACSSnapshot(): END
# =====
```

If an appropriate backup image is found, the meta data is read from the protocol file and the restore is started by invoking the customer library.

### db2ACSEndOperation

The database manager calls db2ACSEndOperation() to end the operation. The return code of 0 indicates that the restore operation was successful.

```
# =====
# db2ACSEndOperation(): BEGIN [2012-11-30 08:27:41]
END_ACTION=0
# db2ACSEndOperation(): END
# =====
```

### db2ACSTerminate

The database manager calls db2ACSTerminate() to terminate the session.

```
# =====
# db2ACSTerminate(): BEGIN [2012-11-30 08:27:41]
# db2ACSTerminate(): END
# =====
```

## Example protocol file for a snapshot deletion

This section contains an example protocol file written for a deletion of a snapshot image which invokes the sample script. During a deletion, the protocol file for the snapshot backup is read, while at the same time new protocol files are written for the delete operation. If the deletion is successful, those protocol files are removed. For illustrative purposes, the following protocol file for the deletion of the snapshot has been broken up into sections for each Db2 ACS API function that is a part of the operation.

### db2ACSInitialize

After loading the library for Db2 ACS and querying the version of the Db2 ACS API, the database manager establishes a Db2 ACS session by calling db2ACSInitialize(). This step is required for all operations. Take care that you do not accidentally delete any images by either specifying the database name or using a unique protocol file repository for each snapshot operation. In the following output, for example, all backups contained in the /home/hotel1nx99/jklauke/ directory are deleted.

```
# db2ACSInitialize(): BEGIN [2012-11-20 09:10:17]
REPOSITORY_PATH=/home/hotel1nx99/jklauke/
EXTERNAL_SCRIPT=/home/hotel1nx99/jklauke/libacssc.sh
HANDLE=1353420617
DB_NAME=*
INSTANCE=*
DBPARTNUM=0
SIGNATURE=SQL10020
EXTERNAL_OPTIONS=/home/hotel1nx99/jklauke/
# db2ACSInitialize(): END
# =====
# db2ACSBeginOperation(): BEGIN [2012-11-20 09:10:17]
OPERATION=delete
# db2ACSBeginOperation(): END
# =====
```

### db2ACSBeginOperation

The database manager calls db2ACSBeginOperation() to begin the specified operation (indicated in the OPERATION flag).

```
# db2ACSBeginOperation(): BEGIN [2012-11-30 08:27:38]
OPERATION=restore
# db2ACSBeginOperation(): END
# =====
```

### db2ACSBeginQuery

The database manager calls db2ACSBeginQuery() to determine which snapshot backup objects are available to be deleted and to prepare the restore. The protocol file also shows the command with which the script was invoked for the prepare action and the return code of the preparation.

```
# =====
# db2ACSBeginQuery(): BEGIN [2012-12-13 08:24:42]
QUERY_TYPE=0x0
QUERY_DBPARTNUM=-1
QUERY_DB=*
QUERY_INSTANCE=*
QUERY_HOST=*
QUERY_OWNER=*
# cmd: /home/hotel1nx99/jklauke/sql1lib/samples/BARVendor/libacssc.sh -a prepare
      -c /home/hotel1nx99/jklauke/repository/db2acs.0.1355405082.cfg
RC_PREPARE=0
# db2ACSBeginQuery(): END [2012-12-13 08:24:42]
# =====
```

### db2ACSGetNextObject

The database manager calls db2ACSGetNextObject() to find an appropriate

backup image for the given timestamp. The function is called iteratively and loops over the available files, giving information about each backup image to the database manager. The following output shows the looping over three protocol files:

```
# =====
# db2ACSGetNextObject(): BEGIN [2012-11-20 09:10:17]
RESULT_0_FILE=/home/hotellnx99/jklauke/db2acs.SAMPLE.0.jklauke.1353420375.cfg
# read result object with timestamp 20121120090616
# db2ACSGetNextObject(): END
# =====
# db2ACSGetNextObject(): BEGIN [2012-11-20 09:10:17]
# db2ACSGetNextObject(): END
# =====
```

### db2ACSDelete

The database manager calls db2ACSDelete() to delete recovery objects. For every backup image that matches the timestamp (retrieved during the db2ACSGetNextObject() call), the API and the script are called sequentially to delete the backup images and any dependent files.

```
# =====
# db2ACSDelete(): BEGIN [2012-12-13 08:24:44]
DELETE_OBJ_ID=5
# cmd: /home/hotellnx99/jklauke/sqllib/samples/BARVendor/libacssc.sh -a delete
# -o 5 -t 20121213051805
# -c /home/hotellnx99/jklauke/repository/db2acs.0.1355405082.cfg
# Delete old backup file and logs:
# /home/hotellnx99/jklauke/repository/SAMPLE.0.jklauke.0.20121213051805.001.tar
# Delete old configuration file:
# /home/hotellnx99/jklauke/repository/db2acs.SAMPLE.0.jklauke.1355393884.cfg
RC_DELETE=0
# db2ACSDelete(): END [2012-12-13 08:24:44]
# =====
```

### db2ACSEndQuery

The database manager calls db2ACSEndQuery() to terminate the query session for backup images.

```
# =====
# db2ACSEndQuery(): BEGIN [2012-11-20 09:10:19]
# db2ACSEndQuery(): END
# =====
```

### db2ACSEndOperation

The database manager calls db2ACSEndOperation() to end the operation. The return code of 0 indicates that the deletion was successful.

```
# =====
# db2ACSEndOperation(): BEGIN [2012-11-20 09:10:19]
END_ACTION=0
# db2ACSEndOperation(): END
# =====
```

### db2ACSTerminate

The database manager calls db2ACSTerminate() to terminate the session.

```
# =====
# db2ACSTerminate(): BEGIN [2012-11-20 09:10:19]
# db2ACSTerminate(): END
# =====
```

## Db2 Advanced Copy Services (ACS) user scripts

By providing your own script for snapshot operations, you can use storage hardware that does not provide a vendor library.

A script specifies the type of snapshot operation that you want performed, as well as some additional options. You specify the script name with the **-script**

parameter for the appropriate command or API. The library for Db2 ACS invokes the script at various times through the operation.

You have to create the script yourself and ensure that it is executable. There is a sample script called `libacssc.sh` provided in `samples/BARVendor` for your reference. The sample script creates one tar file containing the database files and, if logs are included, a second one for the log files. You can use the sample script as a template for your own script, with the appropriate modifications that set it up for your storage device. You would probably want to remove the section that creates the tar file.

## Snapshot backup

During a snapshot backup, the script extracts the information that is required for the current phase from the protocol file and runs the required actions for creating the snapshot. The script writes progress information to the protocol file for debugging reasons.

A snapshot backup script can implement the following actions, preceded by the flag `-a`:

### **prepare**

Runs any actions that need to take place before the snapshot is performed

### **snapshot**

Performs the snapshot

**verify** Verifies that the snapshot was successfully produced (that is, the vendor tools did not return any errors)

### **store\_metadata**

Specifies actions that can occur after the snapshot has been produced and all required metadata has been stored to the protocol file. For example, the script can back up the protocol file or rename the backup image.

### **rollback**

Cleans up the image if a snapshot has failed

## Snapshot restore

During snapshot restores the protocol files that were written during snapshots are read, and new protocol files are written (to the same repository) to show the progress of the restore operation. Every restore operation writes a new protocol file. If the restore is successful, the corresponding protocol file is removed. If the operation fails, the protocol file remains for debugging purposes.

A snapshot restore script can implement the following actions, preceded by the flag `-a`:

### **prepare**

Runs any actions that need to take place before the restore is performed

### **restore**

Restores the snapshot backup image

## Snapshot management

When a snapshot backup image is deleted, the protocol files that were written during snapshots are read, and new protocol files are written (to the same

repository) to show the progress of the delete operation. If the delete operation is successful, the corresponding protocol file is removed. If the operation fails, the protocol file remains for debugging purposes.

A snapshot delete script can implement the following actions, preceded by the flag `-a`:

**prepare**

Runs any actions that need to take place before the restore is performed

**delete** Deletes the snapshot backup image

## Db2 Advanced Copy Services (ACS) API

The Db2 Advanced Copy Services (ACS) application programming interface (API) defines a set of functions that the database manager uses to communicate with storage hardware to perform snapshot backup operations. The Db2 ACS API is only supported on Linux and UNIX operating systems.

To perform snapshot backup and restore operations, you need one of two things:

- A Db2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.
- For storage devices that are not supported, implement a custom script that allows your storage device to perform snapshot operations.

### Db2 Advanced Copy Services (ACS) API functions

The database manager communicates Db2 ACS requests to storage hardware through the Db2 ACS API functions.

#### **db2ACSQueryApiVersion - return the current version of the Db2 Advanced Copy Services (ACS) API:**

Returns the current version of the Db2 Advanced Copy Services (ACS) API.

#### **API include file**

`db2ACSApi.h`

#### **API and data structure syntax**

```
db2ACS_Version db2ACSQueryApiVersion();
```

#### **Parameters**

None.

#### **Usage notes**

Possible return values:

- `DB2ACS_API_VERSION1`
- `DB2ACS_API_VERSION_UNKNOWN`

#### **db2ACSInitialize - initialize a Db2 Advanced Copy Services (ACS) session:**

Initializes a new Db2 Advanced Copy Services (ACS) session. This call establishes communication between the database manager's Db2 ACS library and the Db2 ACS API driver for the storage hardware.

## Include file

db2ACSApi.h

## Syntax and data structures

```
/* =====  
 * Session Initialization  
 * ===== */  
db2ACS_RC db2ACSInitialize(  
    db2ACS_CB          * pControlBlock,  
    db2ACS_ReturnCode * pRC );
```

## Parameters

### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

Before calling db2ACSInitialize(), the database manager populates the following fields:

```
pControlBlock->session  
pControlBlock->options
```

The Db2 ACS API driver populates the following fields before returning:

```
pControlBlock->handle  
pControlBlock->vendorInfo
```

### pRC

Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 23. Return codes

| Return code           | Description                                                                                    | Notes                                                                                                 |
|-----------------------|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK          | The operation was successful.                                                                  |                                                                                                       |
| DB2ACS_RC_INIT_FAILED | The database manager attempted to initialize a Db2 ACS session, but the initialization failed. |                                                                                                       |
| DB2ACS_RC_INV_ACTION  | The database manager requested an action from the Db2 ACS API driver that is invalid.          | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |



Table 23. Return codes (continued)

| Return code              | Description                                                                            | Notes                                                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_COMM_ERROR     | There was a communication error with a storage device, such as a tape drive.           | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_NO_DEV_AVAIL   | There is currently no storage device, such as a tape drive, available to use.          | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

Before the database manager can make any Db2 ACS API calls, except calls to `db2ACSQueryAPIVersion()`, the database manager must call `db2ACSInitialize()`. Once the database manager establishes a Db2 ACS session by calling `db2ACSInitialize()`, then the database manager can perform any combination of Db2 ACS query, read, write, or delete operations. The database manager can terminate the Db2 ACS session by calling `db2ACSTerminate()`.

### db2ACSTerminate - terminate a Db2 Advanced Copy Services (ACS) session:

Terminates a Db2 Advanced Copy Services (ACS) session.

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
/* =====
 * Session Termination
 * ===== */
db2ACS_RC db2ACSTerminate(
    db2ACS_CB          * pControlBlock,
    db2ACS_ReturnCode * pRC );
```

## Parameters

### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

The database manager allocated the memory for this parameter before calling db2ACSInitialize(). The database manager is responsible for freeing this memory after db2ACSTerminate().

Before calling db2ACSTerminate(), the database manager populates the following fields:

pControlBlock->options

The Db2 ACS API driver might invalidate and free the memory in pControlBlock->vendorInfo.vendorCB.

### pRC

Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 24. Return codes

| Return code       | Description                                                                           | Notes                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK      | The operation was successful.                                                         | Free all memory allocated for this session and terminate.                                             |
| DB2ACS_INV_ACTION | The database manager requested an action from the Db2 ACS API driver that is invalid. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

## Usage notes

The Db2 ACS API driver should free all memory that the driver allocated for the Db2 ACS session in `db2ACSTerminate()`.

Regardless of whether `db2ACSTerminate()` completes without error, the database manager cannot call any Db2 ACS functions on this Db2 ACS session again, without first calling `db2ACSInitialize()`.

### **db2ACSPrepare - prepare to perform a snapshot backup operation.:**

When a snapshot backup is performed, the database manager suspends the database. `db2ACSPrepare()` performs all the steps to prepare to perform a snapshot backup operation up to, but not including, the point where the database manager suspends the database.

## Include file

`db2ACSApi.h`

## Syntax and data structures

```
/* =====  
 * Prepare  
 * ===== */  
db2ACS_RC db2ACSPrepare(  
    db2ACS_GroupList * pGroupList,  
    db2ACS_CB * pControlBlock,  
    db2ACS_ReturnCode * pRC );
```

## Parameters

### **pGroupList**

Data type: `db2ACS_GroupList *`

`db2ACS_GroupList` contains a list of groups to be included in the snapshot backup operation.

If **pGroupList** is NULL, all groups (paths) will be included in the snapshot backup operation.

If **pGroupList** is not NULL:

- **pGroupList** contains a list of groups (paths) to be included in the snapshot backup operation.
- The database manager is responsible for allocating and freeing the memory for **pGroupList**.
- The database manager populates the following fields before passing **pGroupList** to `db2ACSPrepare()`:

```
pGroupList->numGroupID  
pGroupList->id
```

### **pControlBlock**

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling `db2ACSPrepare()`, the database manager populates the following fields:

pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options

**pRC** Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 25. Return codes

| Return code              | Description                                                                            | Notes                                                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                          |                                                                                                       |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.  | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

## Usage notes

If db2ACSPrepare() succeeds, then the database manager will suspend the database before calling db2ACSSnapshot().

## db2ACSBEGINOperation - begin a Db2 Advanced Copy Services (ACS) operation.:

Begins a Db2 Advanced Copy Services (ACS) operation.

### Include file

db2ACSApi.h

### Syntax and data structures

```
/* =====  
 * Operation Begin  
 *  
 * A valid ACS operation is specified by passing an ObjectType OR'd with one of  
 * the following Operations, such as:  
 *  
 * (DB2ACS_OP_CREATE | DB2ACS_OBJTYPE_SNAPSHOT)  
 * ===== */  
db2ACS_RC db2ACSBEGINOperation(  
    db2ACS_Operation    operation,  
    db2ACS_CB           * pControlBlock,  
    db2ACS_ReturnCode   * pRC );
```

### Parameters

#### operation

Data type: db2ACS\_Operation.

**operation** is a bitmask indicating the Db2 ACS operation to begin, and the type of object involved.

Operation types:

DB2ACS\_OP\_CREATE  
DB2ACS\_OP\_READ  
DB2ACS\_OP\_DELETE

Object types:

DB2ACS\_OBJTYPE\_BACKUP  
DB2ACS\_OBJTYPE\_LOG  
DB2ACS\_OBJTYPE\_LOADCOPY  
DB2ACS\_OBJTYPE\_SNAPSHOT

For example: ( DB2ACS\_OP\_CREATE | DB2ACS\_OBJTYPE\_SNAPSHOT ) or ( DB2ACS\_OP\_DELETE | DB2ACS\_OBJTYPE\_LOADCOPY ).

The database manager passes **operation** to the db2ACSBEGINOperation() function call.

#### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSBEGINOperation(), the database manager populates the following fields:

pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options

If **operation** is DB2ACS\_OP\_CREATE or DB2ACS\_OP\_READ, then the database manager also populates the following field:

pControlBlock->operation

The information contained within pControlBlock->operation is only valid within the context of a particular Db2 ACS operation. pControlBlock->operation will be set during db2ACSBeginOperation(), and will remain unchanged until db2ACSEndOperation() returns. Neither the database manager nor the Db2 ACS API driver should reference pControlBlock->operation outside the scope of a Db2 ACS operation.

**pRC** Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 26. Return codes

| Return code           | Description                                                                           | Notes                                                                                                 |
|-----------------------|---------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK          | The operation was successful.                                                         |                                                                                                       |
| DB2ACS_RC_INV_OPTIONS | The database manager specified invalid options.                                       |                                                                                                       |
| DB2ACS_RC_INV_ACTION  | The database manager requested an action from the Db2 ACS API driver that is invalid. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

## Usage notes

None.

## db2ACSEndOperation - End a Db2 Advanced Copy Services (ACS) operation:

Ends a Db2 Advanced Copy Services (ACS) operation.

## Include file

db2ACSApi.h

## Syntax and data structures

```
/* =====  
 * Operation End  
 * ===== */  
db2ACS_RC db2ACSEndOperation(  
    db2ACS_EndAction    endAction,  
    db2ACS_CB           * pControlBlock,  
    db2ACS_ReturnCode   * pRC );
```

## Parameters

### endAction

Data type: db2ACS\_EndAction.

**endAction** is a bitmask indicating how the Db2 ACS API driver should end the Db2 ACS operation.

Values:

DB2ACS\_END\_COMMIT  
DB2ACS\_END\_ABORT

The database manager passes **endAction** to the db2ACSEndOperation() function call.

### pControlBlock

Data type: db2ACS\_CB

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSEndOperation(), the database manager populates the following fields:

pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options

### pRC

Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 27. Return codes

| Return code  | Description                   | Notes |
|--------------|-------------------------------|-------|
| DB2ACS_RC_OK | The operation was successful. |       |

Table 27. Return codes (continued)

| Return code             | Description                                                                                   | Notes                                                                                                 |
|-------------------------|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_INV_ACTION    | The database manager requested an action from the Db2 ACS API driver that is invalid.         | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_COMMIT_FAILED | The Db2 ACS API driver could not commit a transaction.                                        |                                                                                                       |
| DB2ACS_RC_ABORT_FAILED  | The database manager attempted to abort a Db2 ACS operation, but the attempt to abort failed. |                                                                                                       |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

If the database manager passes `DB2ACS_END_ABORT` as the **endAction** parameter, the result should be that the snapshot backup objects are deleted.

### db2ACSBeginQuery - begin a query about snapshot backup objects:

Begins a Db2 Advanced Copy Services (ACS) query operation about snapshot backup objects that are available to be used for restore operations.

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
db2ACS_RC db2ACSBeginQuery(
    db2ACS_QueryInput    * pQueryInput,
    db2ACS_CB            * pControlBlock,
    db2ACS_ReturnCode    * pRC );
```

### Parameters

#### pQueryInput

Data type: `db2ACS_QueryInput *`

`db2ACS_QueryInput` has the same fields as `db2ACS_ObjectInfo`.

`db2ACS_ObjectInfo` contains information about object created using the Db2 Advanced Copy Services (ACS) API.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.



Before calling `db2ACSBEGINQUERY()`, the database manager populates the fields of **pQueryInput**.

The Db2 ACS API driver must support the use of the following wildcards in the query:

- `DB2ACS_WILDCARD` in string fields
- `DB2ACS_ANY_PARTITIONNUM` for database partition fields
- `DB2ACS_ANY_UINT32` for 32-bit unsigned integer (UInt32) fields

### **pControlBlock**

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling `db2ACSBEGINQUERY()`, the database manager populates the following fields:

```
pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options
```

**PRC** Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **PRC** before returning.

### **Return Codes**

*Table 28. Return codes*

| Return code                           | Description                                                                            | Notes                                                                                                 |
|---------------------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <code>DB2ACS_RC_OK</code>             | The operation was successful.                                                          |                                                                                                       |
| <code>DB2ACS_RC_INV_ACTION</code>     | The database manager requested an action from the Db2 ACS API driver that is invalid.  | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_INV_DEV_HANDLE</code> | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_DEV_ERROR</code>      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_IO_ERROR</code>       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

`db2ACSBeginQuery()` does not return any query data.

### **db2ACSGetNextObject - list next snapshot backup object available to use for restore:**

Returns the next item in a list of snapshot backup objects that are available to be used for a restore operation.

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
db2ACS_RC db2ACSGetNextObject(
    db2ACS_QueryOutput * pQueryOutput,
    db2ACS_CB          * pControlBlock,
    db2ACS_ReturnCode  * pRC );
```

### Parameters

#### **pQueryOutput**

Data type: `db2ACS_QueryOutput *`

`db2ACS_QueryOutput` contains query result information about snapshot backup objects.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pQueryOutput** before returning.

#### **pControlBlock**

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling `db2ACSGetNextObject()`, the database manager populates the following fields:

```
pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options
```

**pRC** Data type: `db2ACS_ReturnCode *`

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 29. Return codes

| Return code              | Description                                                                                         | Notes                                                                                                                                                                                                           |
|--------------------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                                       |                                                                                                                                                                                                                 |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.               | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session.                                                                                                           |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                                | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session.                                                                                                           |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                                     | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session.                                                                                                           |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations.              | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session.                                                                                                           |
| DB2ACS_RC_OBJ_NOT_FOUND  | The Db2 ACS API driver could not find the snapshot backup object specified by the database manager. | The function call didn't fail, but there are no snapshot backup objects that match the criteria passed to db2ACSBeginQuery().                                                                                   |
| DB2ACS_RC_END_OF_DATA    | The Db2 ACS API driver cannot find any more snapshot backup objects.                                | The function call didn't fail, but there are no more snapshot backup objects that match the criteria passed to db2ACSBeginQuery().                                                                              |
| DB2ACS_RC_MORE_DATA      | There is more data to be transferred from the storage location to the database manager.             | Information about a snapshot backup object that matches the criteria passed to db2ACSBeginQuery() is returned, and there are more snapshot backup objects that match the criteria passed to db2ACSBeginQuery(). |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

The database manager must call `db2ACSBEGINQuery()` before calling `db2ACSGetNextObject()`. The database manager specifies the search criteria in the `db2ACS_QueryInput` parameter passed to `db2ACSBEGINQuery()`.

`db2ACSGetNextObject()` returns information about one snapshot backup object that matches the search criteria passed to `db2ACSBEGINQuery()`. If `db2ACSGetNextObject()` returns `DB2ACS_RC_MORE_DATA`, the database manager can call `db2ACSGetNextObject()` again to receive information about another snapshot backup object that matches the search criteria. If `db2ACSGetNextObject()` returns `DB2ACS_RC_END_OF_DATA`, there are no more snapshot backup objects that match the search criteria.

### **db2ACSEndQuery - end a query about snapshot backup objects:**

The database manager uses the Db2 Advanced Copy Services (ACS) API functions `db2ACSBEGINQuery()` and `db2ACSGetNextObject()` to query about snapshot backup objects that are available to use for restore operations. `db2ACSEndQuery()` terminates that Db2 ACS query session.

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
db2ACS_RC db2ACSEndQuery(  
    db2ACS_CB          * pControlBlock,  
    db2ACS_ReturnCode * pRC );
```

### Parameters

#### **pControlBlock**

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling `db2ACSEndQuery()`, the database manager populates the following fields:

```
pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options
```

**pRC** Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 30. Return codes

| Return code              | Description                                                                            | Notes                                                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                          |                                                                                                       |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.  | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

The database manager cannot call `db2ACSGetNextObject()` again on this Db2 ACS session without first calling `db2ACSBeginQuery()` again.

### db2ACSSnapshot - perform a Db2 Advanced Copy Services (ACS) operation:

Performs a Db2 Advanced Copy Services (ACS) operation.

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
typedef union db2ACS_ReadList
{
    db2ACS_GroupList      group;
} db2ACS_ReadList;

db2ACS_RC db2ACSSnapshot(
    db2ACS_Action          action,
    db2ACS_ObjectID        objectID,
```

```

db2ACS_ReadList      * pReadList,
db2ACS_CB            * pControlBlock,
db2ACS_ReturnCode    * pRC );

```

## Parameters

**action** Data type: db2ACS\_Action

The type of Db2 ACS action to perform. Values:

```

DB2ACS_ACTION_WRITE
DB2ACS_ACTION_READ_BY_OBJECT
DB2ACS_ACTION_READ_BY_GROUP

```

The database manager passes **action** in to db2ACSSnapshot().

## objectID

Data type: db2ACS\_ObjectID

A db2ACS\_ObjectID is a unique identifier for each stored object, which is returned by a query to the storage repository. A db2ACS\_ObjectID is guaranteed to be unique and persistent only within the timeframe of a single Db2 ACS session.

If the database manager specified DB2ACS\_OP\_READ or DB2ACS\_OP\_DELETE as **operation** in the call to db2ACSBeginOperation(), then the database manager passes the value for **objectID** in to db2ACSSnapshot().

## pReadList

Data type: db2ACS\_ReadList \*

db2ACS\_ReadList contains a list of groups.

**pReadList** is only used if **action** is DB2ACS\_ACTION\_READ\_BY\_GROUP.

If **action** is DB2ACS\_ACTION\_READ\_BY\_GROUP, then the database manager is responsible for allocating memory for and populating the fields of **pReadList** before calling db2ACSSnapshot(), and for freeing the memory for **pReadList** afterwards.

## pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSSnapshot(), the database manager populates the following fields:

```

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

```

**pRC** Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 31. Return codes

| Return code              | Description                                                                            | Notes                                                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                          |                                                                                                       |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.  | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

## Usage notes

The database manager calls `db2ACSBeginOperation()` before calling `db2ACSPartition()`, `db2ACSPrepare()`, and then `db2ACSSnapshot()`. The database manager specifies the type of Db2 ACS operation that the Db2 ACS API driver should perform in the **operation** parameter in the call to `db2ACSBeginOperation()`.

## db2ACSPartition - group target data for a database partition together:

Associates a group identifier with each of the paths listed by the database manager as belonging to a database partition.

## Include file

`db2ACSApi.h`

## Syntax and data structures

```
/* =====  
 * Partition  
 * ===== */  
db2ACS_RC db2ACSPartition(  
    db2ACS_PathList    * pPathList,
```

```

db2ACS_CreateObjectInfo * pCreateObjInfo,
db2ACS_CB               * pControlBlock,
db2ACS_ReturnCode       * pRC );

```

## Parameters

### pPathList

Data type: db2ACS\_PathList

db2ACS\_PathList contains a list of database paths, including some extra information about each of those paths specific to Db2 ACS operations.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The **entry** field of the db2ACS\_PathList structure is an array of elements of type db2ACS\_PathEntry. db2ACS\_PathEntry contains information about a database path.

Before calling db2ACSPartition, the database manager populates the following fields of each db2ACS\_PathEntry entry in **pPathList**:

- **path**
- **type**
- **toBeExcluded**

Every path identified by the database manager as belonging to this database partition is given a group identifier by the Db2 ACS API driver. The Db2 ACS API driver populates the **groupID** field of each db2ACS\_PathEntry in **pPathList** before returning.

### pCreateObjInfo

Data type: db2ACS\_CreateObjectInfo

db2ACS\_CreateObjectInfo contains information about the Db2 ACS backup object creation.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The database manager populates the fields of **pCreateObjInfo** before calling db2ACSPartition.

### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSPartition(), the database manager populates the following fields:

```

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

```

### pRC

Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.



The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

*Table 32. Return codes*

| Return code                | Description                                                                                                                       | Notes                                                                                                 |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK               | The operation was successful.                                                                                                     |                                                                                                       |
| DB2ACS_RC_INIT_FAILED      | The database manager attempted to initialize a Db2 ACS session, but the initialization failed.                                    |                                                                                                       |
| DB2ACS_RC_INV_ACTION       | The database manager requested an action from the Db2 ACS API driver that is invalid.                                             | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_INV_DEV_HANDLE   | The database manager passed a storage device handle that is invalid.                                                              | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR        | There was an error with a storage device, such as a tape drive.                                                                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR         | The Db2 ACS API driver encountered an error resulting from input or output operations.                                            | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_OBJ_OUT_OF_SCOPE | The database manager attempted to perform a Db2 ACS operation on a recovery object that is not managed by the Db2 ACS API driver. |                                                                                                       |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

## Usage notes

Db2 Advanced Copy Services handles the data on a single database partition atomically. That is: the data for one database partition is backed up or restored together, and independently of other database partitions - even when the action is part of an operation involving multiple database partitions. `db2ACSPartition` groups database path information for a single database partition together.

The database manager calls `db2ACSPartition` before calling `db2ACSSnapshot`. The database manager will list all the paths associated with this database partition in

the **pPathList** parameter. The database manager can perform a Db2 ACS operation on a subset of the paths listed in **pPathList** by specifying that subset of paths in the **pReadList** parameter passed to db2ACSSnapshot.

**db2ACSVerify - verify that a Db2 Advanced Copy Services (ACS) operation has completed successfully:**

Verifies that a Db2 Advanced Copy Services (ACS) operation succeeded

#### Include file

db2ACSApi.h

#### Syntax and data structures

```
/* =====  
 * Verify  
 * ===== */  
db2ACS_RC db2ACSVerify(  
    db2ACS_PostObjectInfo * pPostObjInfo,  
    db2ACS_CB * pControlBlock,  
    db2ACS_ReturnCode * pRC );
```

#### Parameters

##### pPostObjInfo

Data type: db2ACS\_PostObjectInfo

db2ACS\_DB2ID is a set of data that can not be known at snapshot backup object creation time, but which must be maintained in the object repository.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The database manager populates the fields of **pPostObjInfo** before calling db2ACSVerify. **pPostObjInfo** contains information that is relevant after the Db2 ACS operation. For example, after a successful snapshot backup, **pPostObjInfo** might contain the first active log file. If there is no data relevant for after the Db2 ACS operation, then the database manager will set **pPostObjInfo** to NULL.

##### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSVerify(), the database manager populates the following fields:

```
pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options
```

##### pRC

Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 33. Return codes

| Return code              | Description                                                                            | Notes                                                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                          |                                                                                                       |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.  | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

## Usage notes

If `db2ACSVerify` returns that a snapshot backup operation succeeded, that means that the recovery objects generated by the snapshot backup are available to be used for restore operations.

## **db2ACSDelete - delete recovery objects that were created using Db2 Advanced Copy Services (ACS):**

Deletes recovery objects that were created using Db2 Advanced Copy Services (ACS)

## Include file

`db2ACSApi.h`

## Syntax and data structures

```
/* =====  
 * Delete  
 * ===== */  
db2ACS_RC db2ACSDelete(  
    db2ACS_ObjectID    objectID,  
    db2ACS_CB          * pControlBlock,  
    db2ACS_ReturnCode  * pRC );
```

### Parameters

#### objectID

Data type: db2ACS\_ObjectID

A db2ACS\_ObjectID is a unique identifier for each stored object, which is returned by a query to the storage repository. A db2ACS\_ObjectID is guaranteed to be unique and persistent only within the timeframe of a single Db2 ACS session.

The database manager can use db2ACSQuery() to obtain a valid **objectID** to pass to db2ACSDelete().

#### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSDelete(), the database manager populates the following fields:

```
pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options
```

**pRC** Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

### Return Codes

Table 34. Return codes

| Return code             | Description                                                                                                     | Notes                                                                                                 |
|-------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK            | The operation was successful.                                                                                   | The specified object is deleted. No further Db2 ACS operations can be performed on that object.       |
| DB2ACS_RC_DELETE_FAILED | The Db2 ACS API driver could not successfully delete snapshot backup objects specified by the database manager. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

Table 34. Return codes (continued)

| Return code              | Description                                                                                         | Notes                                                                                                 |
|--------------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                                | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                                     | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations.              | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_OBJ_NOT_FOUND  | The Db2 ACS API driver could not find the snapshot backup object specified by the database manager. |                                                                                                       |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

When the database manager calls `db2ACSDelete`, the Db2 ACS API driver deletes the recovery object identified by **objectID**.

The database manager calls `db2ACSDelete` when a user calls **db2acsutil** with the DELETE parameter.

### **db2ACSStoreMetaData** - store metadata for a recovery object generated using Db2 Advanced Copy Services (ACS):

Stores metadata about a recovery object that was created using Db2 Advanced Copy Services (ACS)

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
db2ACS_RC db2ACSStoreMetaData(
    db2ACS_MetaData      * pMetaData,
    db2ACS_CB            * pControlBlock,
    db2ACS_ReturnCode    * pRC );
```

## Parameters

### pMetaData

Data type: db2ACS\_MetaData

db2ACS\_MetaData stores snapshot backup meta data.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The metadata stored in the **data** field of **pMetaData** is internal to the database manager, and might change over time, so the Db2 ACS API driver just treats this data as a binary stream.

### pControlBlock

Data type: db2ACS\_CB \*

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling db2ACSStoreMetaData(), the database manager populates the following fields:

```
pControlBlock->handle  
pControlBlock->vendorInfo  
pControlBlock->options
```

### pRC

Data type: db2ACS\_ReturnCode \*

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

## Return Codes

Table 35. Return codes

| Return code              | Description                                                                            | Notes                                                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                          |                                                                                                       |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.  | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                   | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_DEV_ERROR      | There was an error with a storage device, such as a tape drive.                        | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| DB2ACS_RC_IO_ERROR       | The Db2 ACS API driver encountered an error resulting from input or output operations. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:

- If a call to `db2ACSBeginQuery()` previously succeeded the database manager can call `db2ACSEndQuery()`
- If a call to `db2ACSBeginOperation()` previously succeeded, the database manager can call `db2ACSEndOperation()`
- If a call to `db2ACSInitialize()` previously succeeded, the database manager can call `db2ACSTerminate()`

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

A snapshot backup operation is comprised of several Db2 ACS API function calls such as: `db2ACSInitialize`, `db2ACSBeginOperation`, `db2ACSPrepare`, and `db2ACSSnapshot`. `db2ACSStoreMetaData` is part of the overall operation too. All of these API calls, including `db2ACSStoreMetaData` must succeed for the snapshot backup operation to succeed. If `db2ACSStoreMetaData` fails, the recovery object that was generated by the Db2 ACS backup operation is unusable.

### **db2ACSRetrieveMetaData - retrieve metadata about a recovery object generated using Db2 Advanced Copy Services (ACS):**

Retrieves metadata about a recovery object that was created using Db2 Advanced Copy Services (ACS)

### Include file

`db2ACSApi.h`

### Syntax and data structures

```
db2ACS_RC db2ACSRetrieveMetaData(  
    db2ACS_MetaData          * pMetaData,  
    db2ACS_ObjectID          objectID,  
    db2ACS_CB                * pControlBlock,  
    db2ACS_ReturnCode        * pRC );
```

### Parameters

#### **pMetaData**

Data type: `db2ACS_MetaData`

`db2ACS_MetaData` stores snapshot backup meta data.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The metadata stored in the **data** field of **pMetaData** is internal to the database manager, and might change over time, so the Db2 ACS API driver just treats this data as a binary stream.

#### **objectID**

Data type: `db2ACS_ObjectID`

A `db2ACS_ObjectID` is a unique identifier for each stored object, which is returned by a query to the storage repository. A `db2ACS_ObjectID` is guaranteed to be unique and persistent only within the timeframe of a single Db2 ACS session.

The database manager can use `db2ACSQuery()` to obtain a valid **objectID** to pass to `db2ACSRetrieveMetaData()`.

#### **pControlBlock**

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a Db2 ACS session.

Before calling `db2ACSRetrieveMetaData()`, the database manager populates the following fields:

```
pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options
```

#### **pRC** Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The Db2 ACS API driver populates the fields of **pRC** before returning.

### **Return Codes**

*Table 36. Return codes*

| Return code                           | Description                                                                                         | Notes                                                                                                 |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <code>DB2ACS_RC_OK</code>             | The operation was successful.                                                                       |                                                                                                       |
| <code>DB2ACS_RC_INV_ACTION</code>     | The database manager requested an action from the Db2 ACS API driver that is invalid.               | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_INV_DEV_HANDLE</code> | The database manager passed a storage device handle that is invalid.                                | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_DEV_ERROR</code>      | There was an error with a storage device, such as a tape drive.                                     | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_IO_ERROR</code>       | The Db2 ACS API driver encountered an error resulting from input or output operations.              | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |
| <code>DB2ACS_RC_OBJ_NOT_FOUND</code>  | The Db2 ACS API driver could not find the snapshot backup object specified by the database manager. | The Db2 ACS API driver encountered an error. The database manager cannot use the Db2 ACS API session. |

If the Db2 ACS API driver encounters an error, the driver might abort a Db2 ACS operation. The Db2 ACS session cannot be used for any action other than the following:



- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about Db2 ACS API return codes, see the topic: “Db2 Advanced Copy Services (ACS) API return codes” on page 499.

### Usage notes

None.

## Db2 Advanced Copy Services (ACS) API data structures

To call Db2 Advanced Copy Services (ACS) API functions, you must use Db2 ACS API data structures.

### db2ACS\_BackupDetails Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_BackupDetails contains information about a snapshot backup operation.

```
/* ----- */
typedef struct db2ACS_BackupDetails
{
    /* A traditional Db2 backup can consist of multiple objects (logical tapes),
     * where each object is uniquely numbered with a non-zero natural number.
     * ----- */
    db2Uint32          sequenceNum;

    char               imageTimestamp[SQLU_TIME_STAMP_LEN + 1];
} db2ACS_BackupDetails;
```

#### sequenceNum

Data type: db2Uint32.

Identifies a backup object by its unique number.

#### imageTimestamp

Data type: char[].

A character string of length SQLU\_TIME\_STAMP\_LEN + 1.

### db2ACS\_CB Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_CB contains fundamental information required to initialize and terminate a Db2 ACS session.

```
/* =====
 * Db2 Backup Adapter Control Block
 * ===== */
typedef struct db2ACS_CB
{
    /* Output: Handle value for this session.
     * ----- */
    db2Uint32          handle;
    db2ACS_VendorInfo  vendorInfo;

    /* Input fields and parameters.
     * ----- */
    db2ACS_SessionInfo session;
    db2ACS_Options     options;
```

```

/* Operation info is optional, possibly NULL, and is only ever valid
 * within the context of an operation (from call to BeginOperation() until
 * the EndOperation() call returns).
 *
 * The operation info will be present during creation or read operations
 * of snapshot and backup objects.
 * ----- */
db2ACS_OperationInfo    * operation;
} db2ACS_CB;

```

#### handle

Data type: db2UInt32.

A handle to reference the Db2 ACS session.

#### vendorInfo

Data type: db2ACS\_VendorInfo.

db2ACS\_VendorInfo contains information about the Db2 ACS API driver.

#### session

Data type: db2ACS\_SessionInfo.

db2ACS\_SessionInfo contains all the information about the Db2 ACS session.

#### options

Data type: db2ACS\_Options.

db2ACS\_Options specifies options to be used for a Db2 ACS operation. This contents of this string is specific to the Db2 ACS API driver.

#### operation

Data type: db2ACS\_OperationInfo \*.

db2ACS\_OperationInfo contains information about a snapshot backup operation.

### db2ACS\_CreateObjectInfo Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_CreateObjectInfo contains information about the Db2 ACS backup object creation.

```

/* =====
 * Object Creation Parameters.
 * ===== */
typedef struct db2ACS_CreateObjectInfo
{
    db2ACS_ObjectInfo    object;
    db2ACS_DB2ID         db2ID;

    /* -----
     * The following fields are optional information for the database manager
     * to use as it sees fit.
     * ----- */

    /* Historically both the size estimate and management
     * class parameters have been used by the TSM client API for traditional
     * backup objects, log archives, and load copies, but not for snapshot
     * backups.
     * ----- */
    db2UInt64            sizeEstimate;
    char                 mgmtClass[DB2ACS_MAX_MGMTCLASS_SZ + 1];

    /* The appOptions is a copy of the iOptions field of flags passed to Db2's
     * db2Backup() API when this execution was initiated. This field will

```

```

    * only contain valid data when creating a backup or snapshot object.
    * ----- */
    db2UInt32          appOptions;
} db2ACS_CreateObjectInfo;

```

**object** Data type: db2ACS\_ObjectInfo

db2ACS\_ObjectInfo contains information about object created using the Db2 Advanced Copy Services (ACS) API.

**db2ID** Data type: db2ACS\_DB2ID

db2ACS\_DB2ID identifies the IBM Data Server.

**sizeEstimate**

Data type: db2UInt64.

An estimate of the size of backup objects being created. This estimate does not apply to log archives, load copies, or snapshot backups objects.

**mgmtClass**

Data type: db2ACS\_MgmtClass.

A character string of length db2ACS\_MAX\_MGMTCLASS\_SZ + 1.

This does not apply to snapshot backup objects.

**appOptions**

Data type: db2UInt32.

A copy of the backup options passed to the backup command that initiated the snapshot backup.

**db2ACS\_DB2ID Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_DB2ID identifies the IBM Data Server.

```

/* =====
 * Db2 Data Server Identifier
 * ===== */
typedef struct db2ACS_DB2ID
{
    db2UInt32          version;
    db2UInt32          release;
    db2UInt32          level;
    char               signature[DB2ACS_SIGNATURE_SZ + 1];
} db2ACS_DB2ID;

```

**version**

Data type: db2UInt32.

Version of IBM Data Server. For example: 9

**release**

Data type: db2UInt32.

Release level of IBM Data Server. For example: 5

**level** Data type: db2UInt32.

Level identifier for the IBM Data Server. For example: 0

**signature**

Data type: char[].

A character string of length DB2ACS\_SIGNATURE\_SZ + 1. For example: "SQL09050"

### **db2ACS\_GroupList Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_GroupList contains a list of groups to be included in the snapshot backup operation.

```
/* =====
 * Snapshot Group List
 *
 * This is an array of size 'numGroupIDs', indicating the set of groups that
 * are to be included in the snapshot operation.
 * ===== */
typedef struct db2ACS_GroupList
{
    db2UInt32          numGroupIDs;
    db2UInt32          * id;
} db2ACS_GroupList;
```

#### **numGroupIDs**

Data type: db2UInt32.

Number of groups in the array **id**.

**id** Data type: db2UInt32 \*.

An array of group identifiers. The groups identified are the groups (or lists of paths) to be included in the snapshot backup operation.

### **db2ACS\_LoadcopyDetails Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_LoadcopyDetails contains information about a load copy operation.

```
/* ----- */
typedef struct db2ACS_LoadcopyDetails
{
    /* Just like the BackupDetails, a Db2 load copy can consist of multiple
     * objects (logical tapes), where each object is uniquely numbered with a
     * non-zero natural number.
     * ----- */
    db2UInt32          sequenceNum;

    char               imageTimestamp[SQLU_TIME_STAMP_LEN + 1];
} db2ACS_LoadcopyDetails;
```

#### **sequenceNum**

Data type: db2UInt32.

Identifies a backup object by its unique number.

#### **imageTimestamp**

Data type: char[].

A character string of length SQLU\_TIME\_STAMP\_LEN + 1

### **db2ACS\_LogDetails Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_LogDetails contains information that identifies a particular database log file.

```
/* ----- */
typedef struct db2ACS_LogDetails
{
    db2UInt32          fileID;
    db2UInt32          chainID;
} db2ACS_LogDetails;
```

**fileID** Data type: db2UInt32.

A number which is the file name of the database log file.

**chainID**

Data type: db2UInt32.

A number which identifies the database log file chain to which the database log file **fileID** belongs.

**db2ACS\_ObjectInfo Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_ObjectInfo contains information about object created using the Db2 Advanced Copy Services (ACS) API.

```
/* =====
 * Object Description and Associated Information.
 *
 * This structure is used for both input and output, and its contents define
 * the minimum information that must be recorded about any object created
 * through this interface.
 * ===== */
typedef struct db2ACS_ObjectInfo
{
    db2ACS_ObjectType      type;
    SQL_PDB_NODE_TYPE     dbPartitionNum;

    char                   db[SQL_DBNAME_SZ + 1];
    char                   instance[DB2ACS_MAX_OWNER_SZ + 1];
    char                   host[SQL_HOSTNAME_SZ + 1];
    char                   owner[DB2ACS_MAX_OWNER_SZ + 1];

    union
    {
        db2ACS_BackupDetails    backup;
        db2ACS_LogDetails       log;
        db2ACS_LoadcopyDetails  loadcopy;
        db2ACS_SnapshotDetails  snapshot;
    } details;
} db2ACS_ObjectInfo;
```

**type** Data type: db2ACS\_ObjectType.

Specifies the snapshot backup objects type. Values:

DB2ACS\_OBJTYPE\_ALL  
DB2ACS\_OBJTYPE\_BACKUP  
DB2ACS\_OBJTYPE\_LOG  
DB2ACS\_OBJTYPE\_LOADCOPY  
DB2ACS\_OBJTYPE\_SNAPSHOT

DB2ACS\_OBJTYPE\_ALL can only be used as a filter for queries. There are no objects of type 0.

**dbPartitionNum**

Data type: SQL\_PDB\_NODE\_TYPE.

An identifier for this database partition.

**db** Data type: char[].

A character string of length SQL\_DBNAME\_SZ + 1.

**instance**

Data type: char[].

A character string of length DB2ACS\_MAX\_OWNER\_SZ + 1.

**host** Data type: char[].

A character string of length SQL\_HOSTNAME\_SZ + 1.

**owner** Data type: char[].

A character string of length DB2ACS\_MAX\_OWNER\_SZ + 1.

#### **details**

##### **backup**

Data type: db2ACS\_BackupDetails

db2ACS\_BackupDetails contains information about a snapshot backup operation.

**log** Data type: db2ACS\_LogDetails

db2ACS\_LogDetails contains information that identifies a particular database log file.

##### **loadcopy**

Data type: db2ACS\_LoadcopyDetails

db2ACS\_LoadcopyDetails contains information about a load copy operation.

##### **snapshot**

Data type: db2ACS\_SnapshotDetails

db2ACS\_SnapshotDetails contains information about a snapshot backup operation.

#### **db2ACS\_ObjectStatus Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_ObjectStatus contains information about the status or progress of a snapshot backup operation, or the status or usability of a snapshot backup object.

```
typedef struct db2ACS_ObjectStatus
{
    /* The total and completed bytes refer only to the ACS snapshot backup
    * itself, not to the progress of any offloaded tape backup.
    *
    * A bytesTotal of 0 indicates that the progress could not be determined.
    * ----- */
    db2UInt64          bytesCompleted;
    db2UInt64          bytesTotal;
    db2ACS_ProgressState progressState;
    db2ACS_UsabilityState usabilityState;
} db2ACS_ObjectStatus;
```

##### **bytesCompleted**

Data type: db2UInt64.

The amount of the snapshot backup that has completed, in bytes.

##### **bytesTotal**

Data type: db2UInt64.

The size of the completed snapshot backup, in bytes.

##### **progressState**

Data type: db2ACS\_ProgressState.

The state of the snapshot backup operation. Values:

DB2ACS\_PSTATE\_UNKNOWN  
 DB2ACS\_PSTATE\_IN\_PROGRESS  
 DB2ACS\_PSTATE\_SUCCESSFUL  
 DB2ACS\_PSTATE\_FAILED

#### **usabilityState**

Data type: db2ACS\_UsabilityState.

The state of the snapshot backup object, how the snapshot backup object can be used. Values:

DB2ACS\_USTATE\_UNKNOWN  
 DB2ACS\_USTATE\_LOCALLY\_MOUNTABLE  
 DB2ACS\_USTATE\_REMOTELY\_MOUNTABLE  
 DB2ACS\_USTATE\_REPETITIVELY\_RESTOREABLE  
 DB2ACS\_USTATE\_DESTRUCTIVELY\_RESTOREABLE  
 DB2ACS\_USTATE\_SWAP\_RESTOREABLE  
 DB2ACS\_USTATE\_PHYSICAL\_PROTECTION  
 DB2ACS\_USTATE\_FULL\_COPY  
 DB2ACS\_USTATE\_DELETED  
 DB2ACS\_USTATE\_FORCED\_MOUNT  
 DB2ACS\_USTATE\_BACKGROUND\_MONITOR\_PENDING  
 DB2ACS\_USTATE\_TAPE\_BACKUP\_PENDING  
 DB2ACS\_USTATE\_TAPE\_BACKUP\_IN\_PROGRESS  
 DB2ACS\_USTATE\_TAPE\_BACKUP\_COMPLETE

#### **db2ACS\_OperationInfo Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_OperationInfo contains information about a snapshot backup operation.

```
/* =====
 * Operation Info
 *
 * The information contained within this structure is only valid within the
 * context of a particular operation. It will be valid at the time
 * BeginOperation() is called, and will remain unchanged until EndOperation()
 * returns, but must not be referenced outside the scope of an operation.
 * ===== */
typedef struct db2ACS_OperationInfo
{
    db2ACS_SyncMode          syncMode;

    /* List of database and backup operation partitions.
     *
     * For details, refer to the db2ACS_PartitionList definition.
     * ----- */
    db2ACS_PartitionList     * dbPartitionList;
} db2ACS_OperationInfo;
```

#### **syncMode**

Data type: db2ACS\_SyncMode.

The level of synchronization between the backup operations on separate database partitions.

Values:

##### **DB2ACS\_SYNC\_NONE**

No synchronization between related operations on multiple database partitions. Used during operations which do not make use of any synchronization between the multiple database partitions.

### DB2ACS\_SYNC\_SERIAL

Used when performing concurrent snapshot backup operations on multiple database partitions. Each database partition will have its input and output (IO) suspended when the snapshot backup operation is issued, and then the IO on the database partitions is resumed serially, not concurrently.

### DB2ACS\_SYNC\_PARALLEL

Performing a snapshot operation on multiple partitions concurrently. Once all database partitions that are involved in the snapshot backup operation have completed preparations for the snapshot backup operation, input and output (IO) will be suspended on all of the database partitions. The remaining snapshot backup steps will take place concurrently on all of the involved database partitions.

### dbPartitionList

Data type: db2ACS\_PartitionList \*.

db2ACS\_PartitionList contains information about the database partitions that are in the database and that are involved in a Db2 ACS operation.

### db2ACS\_Options Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_Options specifies options to be used for a Db2 ACS operation. This contents of this string is specific to the Db2 ACS API driver.

```
/* =====  
 * Db2 Backup Adapter User Options  
 * ===== */  
typedef struct db2ACS_Options  
{  
    db2UInt32          size;  
    void               * data;  
} db2ACS_Options;
```

**size** Data type: db2UInt32.

Size of **data**, in bytes.

**data** Data type: void \*.

Pointer to a block of memory that contains the options.

### db2ACS\_PartitionEntry Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_PartitionEntry is an element of a db2ACS\_PartitionList.

```
typedef struct db2ACS_PartitionEntry  
{  
    SQL_PDB_NODE_TYPE num;  
    char               host[SQL_HOSTNAME_SZ + 1];  
} db2ACS_PartitionEntry;
```

**num** Data type: SQL\_PDB\_NODE\_TYPE.

An identifier for this database partition entry.

**host** Data type: char[].

A character string of length SQL\_HOSTNAME\_SZ + 1.



### **db2ACS\_PartitionList Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_PartitionList contains information about the database partitions that are in the database and that are involved in a Db2 ACS operation.

```
typedef struct db2ACS_PartitionList
{
    db2UInt64          numPartsInDB;
    db2UInt64          numPartsInOperation;
    db2ACS_PartitionEntry * partition;
} db2ACS_PartitionList;
```

#### **numPartsInDB**

Data type: db2UInt64.

The number of database partitions in the database.

#### **numPartsInOperation**

Data type: db2UInt64.

The number of database partitions involved in the Db2 ACS operation.

#### **partition**

Data type: db2ACS\_PartitionEntry \*.

db2ACS\_PartitionEntry is an element of a db2ACS\_PartitionList.

### **db2ACS\_PathEntry Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_PathEntry contains information about a database path.

```
typedef struct db2ACS_PathEntry
{
    /* INPUT: The path and type will be provided by the database server, as well
     *        as a flag indicating if the path is to be excluded from the backup.
     * ----- */
    char          path[DB2ACS_MAX_PATH_SZ + 1];
    db2ACS_PathType type;
    db2UInt32     toBeExcluded;

    /* OUTPUT: The group ID is to be provided by the backup adapter for use by
     *        the Db2 server. The group ID will be used during with snapshot
     *        operations as an indication of which paths are dependent and must
     *        be included together in any snapshot operation. Unique group IDs
     *        indicate that the paths in those groups are independent for the
     *        purposes of snapshot operations.
     * ----- */
    db2UInt32     groupID;
} db2ACS_PathEntry;
```

**path** Data type: char[].

A character string of length DB2ACS\_MAX\_PATH\_SZ + 1.

**type** Data type: db2ACS\_PathType.

The type of path. Values:

```
DB2ACS_PATH_TYPE_UNKNOWN
DB2ACS_PATH_TYPE_LOCAL_DB_DIRECTORY
DB2ACS_PATH_TYPE_DBPATH
DB2ACS_PATH_TYPE_DB_STORAGE_PATH
DB2ACS_PATH_TYPE_TBSP_CONTAINER
DB2ACS_PATH_TYPE_TBSP_DIRECTORY
DB2ACS_PATH_TYPE_TBSP_DEVICE
```

DB2ACS\_PATH\_TYPE\_LOGPATH  
DB2ACS\_PATH\_TYPE\_MIRRORLOGPATH

**toBeExcluded**

Data type: db2Uint32.

A flag indicating whether to include the given path in the snapshot backup. Values:

- 0 - include the path in the snapshot backup
- 1 - do not include the path in the snapshot backup

**groupID**

Data type: db2Uint32.

A group identifier.

**db2ACS\_PathList Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_PathList contains a list of database paths, including some extra information about each of those paths specific to Db2 ACS operations.

```
/* =====  
 * Snapshot File List  
 *  
 * This is an array of 'numEntries' db2ACS_PathEntry's, where each path entry is  
 * a path to some storage on the Db2 server which is in use by the current  
 * database.  
 * ===== */
```

```
typedef struct db2ACS_PathList  
{  
    db2Uint32          numEntries;  
    db2ACS_PathEntry  * entry;  
} db2ACS_PathList;
```

**numEntries**

Data type: db2Uint32.

The number of path entries in the **entry** array.

**entry** Data type: db2ACS\_PathEntry.

db2ACS\_PathEntry contains information about a database path.

**db2ACS\_PostObjectInfo Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_DB2ID is a set of data that can not be known at snapshot backup object creation time, but which must be maintained in the object repository.

```
/* =====  
 * The PostObjectInfo is a set of data that can not be known at object  
 * creation time, but which must be maintained in the object repository. This  
 * is an optional field on the Verify() call, which may be NULL if there are  
 * no post-operation updates to be made.  
 * ===== */
```

```
typedef struct db2ACS_PostObjectInfo  
{  
    /* The first active log will only be valid when creating a backup or  
    * snapshot object. It will indicate the file number and chain id of the  
    * first log required for recovery using this object.  
    * ----- */  
    db2ACS_LogDetails  firstActiveLog;  
} db2ACS_PostObjectInfo;
```

### **firstActiveLog**

Data type: db2ACS\_LogDetails.

db2ACS\_LogDetails contains information that identifies a particular database log file.

### **db2ACS\_QueryInput and db2ACS\_QueryOutput Db2 Advanced Copy Services (ACS) API data structures:**

db2ACS\_QueryInput contains identifying information for an object about which you are querying. db2ACS\_QueryOutput contains query result information about snapshot backup objects.

```
/* =====
 * Unique Querying.
 *
 * When using this structure as query input, to indicate the
 * intention to supply a 'wildcard' search criteria, Db2 will supply:
 *
 * -- character strings as "*".
 * -- numeric values as (-1), cast as the appropriate signed or unsigned
 * type.
 * ===== */

typedef struct db2ACS_ObjectInfo db2ACS_QueryInput;

typedef struct db2ACS_QueryOutput
{
    db2ACS_ObjectID      objectID;
    db2ACS_ObjectInfo    object;
    db2ACS_PostObjectInfo postInfo;
    db2ACS_DB2ID         db2ID;
    db2ACS_ObjectStatus  status;

    /* Size of the object in bytes.
     * ----- */
    db2UInt64            objectSize;

    /* Size of the metadata associated with the object, if any, in bytes.
     * ----- */
    db2UInt64            metaDataSize;

    /* The creation time of the object is a 64bit value with a definition
     * equivalent to an ANSI C time_t value (seconds since the epoch, GMT).
     *
     * This field is equivalent to the file creation or modification time in
     * a traditional filesystem. This should be created and stored
     * automatically by the BA subsystem, and a valid time value should be
     * returned with object query results, for all object types.
     * ----- */
    db2UInt64            createTime;
} db2ACS_QueryOutput;
```

### **objectID**

Data type: db2ACS\_ObjectID.

A db2ACS\_ObjectID is a unique identifier for each stored object, which is returned by a query to the storage repository. A db2ACS\_ObjectID is guaranteed to be unique and persistent only within the timeframe of a single Db2 ACS session.

### **object** Data type: db2ACS\_ObjectInfo

db2ACS\_ObjectInfo contains information about object created using the Db2 Advanced Copy Services (ACS) API.

**postInfo**

Data type: db2ACS\_PostObjectInfo.

db2ACS\_DB2ID is a set of data that can not be known at snapshot backup object creation time, but which must be maintained in the object repository.

**db2ID** Data type: db2ACS\_DB2ID.

db2ACS\_DB2ID identifies the IBM Data Server.

**status** Data type: db2ACS\_ObjectStatus.

db2ACS\_ObjectStatus contains information about the status or progress of a snapshot backup operation, or the status or usability of a snapshot backup object.

**objectSize**

Data type: db2UInt64.

Size of the object in bytes.

**metaDataSize**

Data type: db2UInt64.

Size of the metadata associated with the object, if any, in bytes.

**createTime**

Data type: db2UInt64.

The creation time of an object. The value of **createTime** is equivalent to an ANSI C time\_t value.

**db2ACS\_ReadList Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_ReadList contains a list of groups.

```
/* The ReadList will only be used for snapshots where the action is READ, and
 * where one of the granularity modifiers other than BY_OBJ has been specified.
 * In the typical usage scenario of ( READ | BY_OBJ ) the ReadList parameter
 * should be ignored.
 *
 * When the action is DB2ACS_ACTION_BY_GROUP the union is to be interpreted
 * as a group list.
 * ----- */
```

```
typedef union db2ACS_ReadList
```

```
{
    db2ACS_GroupList      group;
} db2ACS_ReadList;
```

**group** Data type: db2ACS\_GroupList.

db2ACS\_GroupList contains a list of groups to be included in the snapshot backup operation.

**db2ACS\_ReturnCode Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS\_ReturnCode parameter for a Db2 ACS API function call will be recorded in the database manager diagnostic logs.

```
/* =====
 * Storage Adapter Return Code and Diagnostic Data.
 *
 * These will be recorded in the Db2 diagnostic logs, but are intended to be
 * internal return and reason codes from the storage layers which can be used
 * in conjunction with the DB2ACS_RC to provide more detailed diagnostic info.
```

```

/* ===== */
typedef struct db2ACS_ReturnCode
{
    int          returnCode;
    int          reasonCode;
    char         description[DB2ACS_MAX_COMMENT_SZ + 1];
} db2ACS_ReturnCode;

```

#### **returnCode**

Data type: int.

Return code specific to the storage hardware.

#### **reasonCode**

Data type: int.

Reason code specific to the storage hardware.

#### **description**

Data type: char[].

A character string of length DB2ACS\_MAX\_COMMENT\_SZ + 1.

### **db2ACS\_SessionInfo Db2 Advanced Copy Services (ACS) API data structure:**

db2ACS\_SessionInfo contains all the information about the Db2 ACS session.

```

/* =====
 * Session Info
 * ===== */
typedef struct db2ACS_SessionInfo
{
    db2ACS_DB2ID          db2ID;

    /* Fields identifying the backup session originator.
     * ----- */
    SQL_PDB_NODE_TYPE     dbPartitionNum;
    char                  db[SQL_DBNAME_SZ + 1];
    char                  instance[DB2ACS_MAX_OWNER_SZ + 1];
    char                  host[SQL_HOSTNAME_SZ + 1];
    char                  user[DB2ACS_MAX_OWNER_SZ + 1];
    char                  password[DB2ACS_MAX_PASSWORD_SZ + 1];

    /* The fully qualified ACS vendor library name to be used.
     * ----- */
    char                  libraryName[DB2ACS_MAX_PATH_SZ + 1];
} db2ACS_SessionInfo;

```

**db2ID** Data type: db2ACS\_DB2ID

db2ACS\_DB2ID identifies the IBM Data Server.

#### **dbPartitionNum**

Data type: SQL\_PDB\_NODE\_TYPE

The unique, numeric identifier for a database partition.

**db** Data type: char[].

A character string of length SQL\_DBNAME\_SZ + 1.

#### **instance**

Data type: char[].

A character string of length DB2ACS\_MAX\_OWNER\_SZ + 1.

**host** Data type: char[].

A character string of length SQL\_HOSTNAME\_SZ + 1.

**user** Data type: char[].

A character string of length DB2ACS\_MAX\_OWNER\_SZ + 1.

**password**

Data type: char[].

A character string of length DB2ACS\_MAX\_PASSWORD\_SZ + 1.

**libraryName**

Data type: char[].

A character string of length DB2ACS\_MAX\_PATH\_SZ + 1.

**db2ACS\_SnapshotDetails** Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_SnapshotDetails contains information about a snapshot backup operation.

```
typedef struct db2ACS_SnapshotDetails
{
    char                imageTimestamp[SQLU_TIME_STAMP_LEN + 1];
} db2ACS_SnapshotDetails;
```

**imageTimestamp**

Data type: char[].

A character string of length SQLU\_TIME\_STAMP\_LEN + 1.

**db2ACS\_MetaData** Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_MetaData stores snapshot backup meta data.

```
/* =====
 * The metadata structure itself is internal to Db2 and is to be treated by
 * the storage interface as an unstructured block of data of the given size.
 * ===== */
typedef struct db2ACS_MetaData
{
    db2UInt64            size;
    void                * data;
} db2ACS_MetaData;
```

**size** Data type: db2UInt32.

Size of **data**, in bytes.

**data** Data type: void \*.

A pointer to a block of memory that the database manager uses to store snapshot backup metadata.

**db2ACS\_VendorInfo** Db2 Advanced Copy Services (ACS) API data structure:

db2ACS\_VendorInfo contains information about the Db2 ACS API driver.

```
/* =====
 * Storage Vendor Identifier
 * ===== */
typedef struct db2ACS_VendorInfo
{
    void                * vendorCB;           /* Vendor control block */
    db2UInt32            version;             /* Current version      */
    db2UInt32            release;            /* Current release      */
    db2UInt32            level;              /* Current level        */
    char                signature[DB2ACS_MAX_VENDORID_SZ + 1];
} db2ACS_VendorInfo;
```

**vendorCB**

Data type: void \*.

Pointer to a control block that is specific to the Db2 ACS API driver.

**version**

Data type: db2UInt32.

Version of the Db2 ACS API driver.

**release**

Data type: db2UInt32.

Release level of the Db2 ACS API driver.

**level**

Data type: db2UInt32.

Level identifier for the Db2 ACS API driver.

**signature**

Data type: db2ACS\_VendorSignature.

A character string of length DB2ACS\_MAX\_VENDORID\_SZ + 1.

**Db2 Advanced Copy Services (ACS) API return codes**

Db2 Advanced Copy Services (ACS) API functions return a defined set of possible return codes.

*Table 37. Db2 Advanced Copy Services (ACS) API return codes*

| Return code              | Description                                                                                                                     |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_OK             | The operation was successful.                                                                                                   |
| DB2ACS_RC_LINK_EXIST     | The session was previously activated.                                                                                           |
| DB2ACS_RC_COMM_ERROR     | There was a communication error with a storage device, such as a tape drive.                                                    |
| DB2ACS_RC_INV_VERSION    | The version of the database manager's Db2 ACS library and the version of the Db2 ACS API driver are not compatible.             |
| DB2ACS_RC_INV_ACTION     | The database manager requested an action from the Db2 ACS API driver that is invalid.                                           |
| DB2ACS_RC_NO_DEV_AVAIL   | There is currently no storage device, such as a tape drive, available to use.                                                   |
| DB2ACS_RC_OBJ_NOT_FOUND  | The Db2 ACS API driver could not find the snapshot backup object specified by the database manager.                             |
| DB2ACS_RC_OBJJS_FOUND    | The Db2 ACS API driver found more than one snapshot backup object that matches the specification given by the database manager. |
| DB2ACS_RC_INV_USERID     | The database manager passed an invalid user id to the Db2 ACS API driver.                                                       |
| DB2ACS_RC_INV_PASSWORD   | The database manager passed an invalid password to the Db2 ACS API driver.                                                      |
| DB2ACS_RC_INV_OPTIONS    | The database manager specified invalid options.                                                                                 |
| DB2ACS_RC_INIT_FAILED    | The database manager attempted to initialize a Db2 ACS session, but the initialization failed.                                  |
| DB2ACS_RC_INV_DEV_HANDLE | The database manager passed a storage device handle that is invalid.                                                            |
| DB2ACS_RC_BUFF_SIZE      | The database manager specified a buffer size that is invalid.                                                                   |

Table 37. Db2 Advanced Copy Services (ACS) API return codes (continued)

| Return code                    | Description                                                                                                                                                                                                                      |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DB2ACS_RC_END_OF_DATA          | The Db2 ACS API driver cannot find any more snapshot backup objects.                                                                                                                                                             |
| DB2ACS_RC_END_OF_TAPE          | The storage device unexpectedly reached the end of tape backup media.                                                                                                                                                            |
| DB2ACS_RC_DATA_RESEND          | A storage device, such as a tape drive, requested that the database manager resend the most recent buffer of data.                                                                                                               |
| DB2ACS_RC_COMMIT_FAILED        | The Db2 ACS API driver could not commit a transaction.                                                                                                                                                                           |
| DB2ACS_RC_DEV_ERROR            | There was an error with a storage device, such as a tape drive.                                                                                                                                                                  |
| DB2ACS_RC_WARNING              | The storage hardware returned a warning. Look in the database manager diagnostic logs for more information.                                                                                                                      |
| DB2ACS_RC_LINK_NOT_EXIST       | The session was not activated previously.                                                                                                                                                                                        |
| DB2ACS_RC_MORE_DATA            | There is more data to be transferred from the storage location to the database manager.                                                                                                                                          |
| DB2ACS_RC_ENDOFMEDIA_NO_DATA   | The storage device reached the end of the storage media without finding any data.                                                                                                                                                |
| DB2ACS_RC_ENDOFMEDIA           | The storage device reached the end of the storage media.                                                                                                                                                                         |
| DB2ACS_RC_MAX_LINK_GRANT       | The maximum number of links has been established. The database manager cannot establish more links.                                                                                                                              |
| DB2ACS_RC_IO_ERROR             | The Db2 ACS API driver encountered an error resulting from input or output operations.                                                                                                                                           |
| DB2ACS_RC_DELETE_FAILED        | The Db2 ACS API driver could not successfully delete snapshot backup objects specified by the database manager.                                                                                                                  |
| DB2ACS_RC_INV_BKUP_FNAME       | The database manager specified an invalid filename for the snapshot backup object.                                                                                                                                               |
| DB2ACS_RC_NOT_ENOUGH_SPACE     | The Db2 ACS API driver estimated that there is not enough storage space to perform a snapshot backup of the database specified by the database manager.                                                                          |
| DB2ACS_RC_ABORT_FAILED         | The database manager attempted to abort a Db2 ACS operation, but the attempt to abort failed.                                                                                                                                    |
| DB2ACS_RC_UNEXPECTED_ERROR     | The Db2 ACS API driver encountered a severe, unknown error.                                                                                                                                                                      |
| DB2ACS_RC_NO_DATA              | The Db2 ACS API driver did not return any data to the database manager.                                                                                                                                                          |
| DB2ACS_RC_OBJ_OUT_OF_SCOPE     | The database manager attempted to perform a Db2 ACS operation on a recovery object that is not managed by the Db2 ACS API driver.                                                                                                |
| DB2ACS_RC_INV_CALL_SEQUENCE    | The database manager made calls to Db2 ACS API functions in a sequence that is invalid. For example, the database manager must call db2ACSInitialize before calling any other Db2 ACS API function except db2ACSQueryAPIVersion. |
| DB2ACS_RC_SHARED_STORAGE_GROUP | The database manager attempted to perform a snapshot operation against a storage object that is being used by another database or application.                                                                                   |



## Return codes for customer scripts

The db2ACS\_ReturnCode data structure contains diagnostic information (specifically a return code and message text) that help you determine why an error occurred and what to do about it. This information is written to the db2diag.log file. You should consult the db2diag.log file because the reason code returned with the standard message might not be accurate. Table 38 contains a list of the return codes that you might see for snapshot operations using a custom script.

*Table 38. Error codes for script-initiated snapshot operations*

| Return code | Error condition                                                              | User response                                                                                                                        |
|-------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| 1           | Memory allocation failed                                                     | Free up memory on the system.                                                                                                        |
| 2           | Options missing                                                              | Provide at least the name of the script to the library.                                                                              |
| 3           | Key not found                                                                | The backup image presented by the protocol file is no longer usable. Move the protocol file away.                                    |
| 4           | Checksum mismatch                                                            | The backup image presented by the protocol file is no longer usable. Move the protocol file away                                     |
| 5           | Meta data read failed                                                        | The backup image presented by the protocol file is no longer usable. Move the protocol file away                                     |
| 6           | Reading of one protocol file failed                                          | Ensure that the userid issuing the operation is allowed to read the protocol file.                                                   |
| 7           | Writing to the protocol file failed                                          | Ensure that there is enough free space available in the file system, and check other circumstances that can cause the flush to fail. |
| 8           | Repository check failed                                                      | Ensure that the repository path exists and the userid issuing the operation can read and write to this path.                         |
| 9           | Script check failed                                                          | Ensure that the script exists and the userid issuing the operation is allowed to read and execute the script.                        |
| 10          | Opening of a protocol file candidate during query, restore, or delete failed | Check the privileges of all files, particularly the one reported in the db2diag.log file.                                            |
| 11          | Closing of protocol file failed                                              | Ensure that the file system of the protocol file repository has enough space.                                                        |
| 12          | Pipe open failed                                                             | Ensure that the operating system has enough resources.                                                                               |

*Table 38. Error codes for script-initiated snapshot operations (continued)*

| <b>Return code</b> | <b>Error condition</b>               | <b>User response</b>                                                                        |
|--------------------|--------------------------------------|---------------------------------------------------------------------------------------------|
| 13                 | Pipe close failed                    | Ensure that the operating system has enough resources.                                      |
| 14                 | Script failed with given return code | Check the return code of the script; this information can be found in the db2diag.log file. |
| 15                 | Wrong object type                    | Rerun the operation.                                                                        |
| 16                 | Reading of the options file failed   | Check the options file.                                                                     |
| 17                 | Closing of the options file failed   | Check the operating system.                                                                 |
| 18                 | Wrong query type                     | Rerun the operation.                                                                        |
| 19                 | Option length too long               | Reduce the number of options that you specify.                                              |

---

# Index

## A

- administration notification log
  - database restart operations 341
  - details 243
- AIX
  - backups 277
  - restores 277
- ALTER DATABASE statement
  - compatibility with online backups 321
- ALTER STOGROUP statement
  - compatibility with online backups 321
- alternate servers
  - examples 33
  - identifying 30
- archive logging
  - configuration parameters 171
  - overview 24
- archiving
  - log files
    - compression 275, 276
    - on demand 165
    - overview 158
    - to tape 165
- assisted remote catchup state 188
- ASYNCR synchronization mode 134
- automated restarts
  - overview 254
- automatic backups
  - enabling 312
  - sample configuration 145
- automatic client reroute
  - alternate servers 30
  - connection failures 29
  - details 26
  - examples 33
  - high availability disaster recovery 115
  - high availability disaster recovery (HADR) 198
  - limitations 31
  - roadmap 26
  - setup 26
- automatic incremental restore
  - limitations 364
- automatic maintenance
  - AUTOMAINT\_SET\_POLICY
    - procedure 144
  - AUTOMAINT\_SET\_POLICYFILE
    - procedure 144
  - backups 269, 312
  - configuring 144
  - policy specification sample 145
- automatic reorganization
  - configuration sample 145
- automatic restart
  - crash recovery 341
- automatic statistics collection
  - configuration sample 145

- autorestart database configuration
  - parameter 118

## B

- backup
  - statistics 320
- BACKUP DATABASE command
  - backing up data 303
  - Db2 pureScale environments 313
- backup images 173, 300
- backup utility
  - authorities required 321
  - displaying information 300
  - examples 323
  - monitoring progress 318
  - overview 300
  - performance 319
  - privileges required 321
  - restrictions 303
  - troubleshooting 300
- backups
  - automatic 269, 312
  - CLP examples 323
  - compression 275
  - databases
    - automatic 269, 312
  - displaying information 300
  - frequency 272
  - incremental 361
  - named pipes 309
  - offline 272
  - online 272
  - operating system restrictions 277
  - partitioned databases 310
  - storage considerations 274
  - tape 307
  - user exit program 274
- blk\_log\_dsk\_ful configuration parameter
  - overview 147
- built-in views
  - DB\_HISTORY
    - viewing recovery history file entries 287

## C

- cascading assignment 8
- circular logging 23, 171
- clients
  - communication errors 26
- clone databases
  - creating
    - using different storage group paths 388
    - using split mirror 100, 101
- cluster caching facilities
  - failover 254
  - restarting 254

- cluster domains
  - database partitions 52
  - mount points 52
  - networks 49
  - overview 47
  - paths 52
- clustering
  - heartbeat monitoring 7
  - IP address takeover 7
- clusters
  - HACMP 8
  - IBM PowerHA SystemMirror for AIX 8
  - managing
    - high availability disaster recovery (HADR) 133
    - resource groups 48
    - resources 48
    - software 7, 47
- command line processor (CLP)
  - examples
    - backing up 323
    - database rebuild sessions 401
    - redirected restore sessions 381
    - rollforward sessions 432
- commands
  - db2adutl
    - cross-node recovery examples 326
    - upload examples 295
- compression
  - backup 275
- configuration
  - databases
    - HADR 127
  - fault monitor
    - db2fm command 37
    - db2fmcu command 38
    - registry file 36
    - high availability 106, 118
- configuration parameters
  - auto\_del\_rec\_obj 294
  - autorestart 341
  - database logging 146, 147
  - hadr\_peer\_window
    - setting 127
  - hadr\_timeout
    - setting 127
  - logarchopt1
    - cross-node recovery examples 326
  - vendoropt
    - cross-node recovery examples 326
- connections
  - failures
    - automatic client reroute 29
    - parameter setting 127
- continuous availability
  - Solaris Operating System cluster support 17
- crash recovery 352
  - details 341
  - initiating 265

- crash recovery (*continued*)
  - member 254
- CREATE STOGROUP statement
  - compatibility with online backups 321
- cross-node database recovery
  - examples 326

## D

- data
  - parity striping by sectors (RAID level 5) 344
  - recovering
    - overview 269
- data recovery
  - log replay delay 194
- database objects
  - recovery history file 269
  - recovery log file 269
  - table space change history file 269
- database partition servers
  - failed 346
  - recovering from failure 350
- database partitions
  - synchronizing clocks 161
- database seeds 373
- databases
  - activating in high availability disaster recovery (HADR) environment 177
  - backups
    - automated 312
    - strategy 269
  - configuring
    - high availability disaster recovery (HADR) 127
  - connections
    - high availability disaster recovery (HADR) 127
  - deactivating
    - high availability disaster recovery (HADR) environment 177
  - logging
    - circular 23
    - configuration parameters 147
    - overview 23
  - nonrecoverable 269
  - rebuilding
    - examples 401
    - incremental backup images 398
    - overview 389
    - partitioned databases 399
    - restrictions 400
    - table space containers 393
    - target image selection 394
  - recovering
    - strategy 269
  - rollforward recovery
    - overview 358
  - temporary table spaces 393
  - transporting schemas
    - examples 414
    - overview 411
    - transportable objects 413
    - troubleshooting 417
- daylight savings time 184

- DB\_HISTORY administrative view
  - viewing recovery history file
    - entries 287
- Db2 Advanced Copy Services (ACS)
  - activating 443
  - best practices 441
  - configuring 444
  - directory 444
  - enabling 442
  - installing
    - process 442
  - overview 440
  - protocol file
    - description 447
    - usage 447
  - restrictions 441
  - scripted interface 446
  - setup\_db2.sh setup script 445
  - user scripts
    - description 457
    - usage 457
- Db2 Advanced Copy Services (ACS) API
  - data structures
    - db2ACS\_BackupDetails 485
    - db2ACS\_CB 485
    - db2ACS\_CreateObjectInfo 486
    - db2ACS\_DB2ID 487
    - db2ACS\_GroupList 488
    - db2ACS\_LoadcopyDetails 488
    - db2ACS\_LogDetails 488
    - db2ACS\_MetaData 498
    - db2ACS\_ObjectInfo 489
    - db2ACS\_ObjectStatus 490
    - db2ACS\_OperationInfo 491
    - db2ACS\_Options 492
    - db2ACS\_PartitionEntry 492
    - db2ACS\_PartitionList 493
    - db2ACS\_PathEntry 493
    - db2ACS\_PathList 494
    - db2ACS\_PostObjectInfo 494
    - db2ACS\_QueryInput 495
    - db2ACS\_QueryOutput 495
    - db2ACS\_ReadList 496
    - db2ACS\_ReturnCode 496
    - db2ACS\_SessionInfo 497
    - db2ACS\_SnapshotDetails 498
    - db2ACS\_VendorInfo 498
    - overview 485
  - functions
    - db2ACSBeginOperation 465
    - db2ACSBeginQuery 468
    - db2ACSDelete 479
    - db2ACSEndOperation 467
    - db2ACSEndQuery 472
    - db2ACSGetNextObject 470
    - db2ACSInitialize 460
    - db2ACSPartition 475
    - db2ACSPrepare 463
    - db2ACSQueryApiVersion 459
    - db2ACSRetrieveMetaData 483
    - db2ACSSnapshot 473
    - db2ACSStoreMetaData 481
    - db2ACSTerminate 461
    - db2ACSVerify 478
    - overview 459
  - return codes 499

- Db2 cluster services
  - overview 253
- Db2 High Availability Feature
  - cluster configuration 44
  - cluster manager
    - integration 42
  - overview 42
- Db2 high availability instance
  - configuration utility
    - see db2haicu utility 53
- Db2 idle processes
  - details 256
- Db2 pureScale
  - HADR
    - preferred replay member 223
    - replay member 223
    - standby replay 223
- Db2 pureScale environments
  - automated restarts 254
  - backups 313
  - database rollforward 427
  - HADR
    - adding members 225
    - overview 218
    - preferred replay member 224
    - removing members 226
    - topology changes 225, 226
  - log file management 279
  - log record identifiers (LRIs) 283
  - log sequence numbers (LSNs) 283
  - log stream merges 279
  - log streams 279
  - restarting 254
  - restoring 313
- DB2\_PEER\_WAIT\_LIMIT registry variable
  - high availability disaster recovery (HADR) 118
- db2adutl command
  - cross-node recovery examples 326
- db2Backup API
  - backing up data 303
- db2fm command
  - fault monitor overview 35
- db2haicu utility
  - cluster domains
    - creating 87
    - maintaining 88
    - overview 47
  - clustered environment 45
  - details 53
  - detected database paths 87
  - input file samples
    - db2ha\_sample\_DPF\_mutual.xml 79
    - db2ha\_sample\_DPF\_NPlusM.xml 82
    - db2ha\_sample\_HADR.xml 84
    - db2ha\_sample\_sharedstorage\_mutual.xml 77
  - input file XML schema
    - ClusterDomainType 60
    - ClusterNodeType 66
    - CustomPolicyType 73
    - DB2PartitionSetType 68
    - DB2PartitionType 69
    - details 57
    - FailoverPolicyType 67
    - HADBDefn 76
    - HADBType 76
    - HADRDBDefn 75

- db2haicu utility *(continued)*
  - input file XML schema *(continued)*
    - HADRDBType 73
    - InterfaceType 64
    - IPAddressType 65
    - MountType 71
    - MutualPolicyType 72
    - NPlusMPolicyType 72
    - PhysicalNetworkType 63
    - QuorumType 61
  - maintenance mode 55
  - prerequisites 86
  - quorum devices 49
  - restrictions 89
  - running
    - interactive mode 56
    - XML input file 56, 77
  - startup mode 55
  - troubleshooting 89
- db2inidb command
  - creating split mirror 104, 105
  - overview 93
- db2pd command
  - HADR standby database states 192
- db2Recover API
  - recovering data 325
- db2Restore API
  - recovering data 368
- db2Rollforward API
  - applying transactions to restored
    - backup image 421
- db2tapemgr command
  - archiving log files to tape 165
- db2uext2 program
  - calling format 169
  - details 167
- disaster recovery
  - high availability disaster recovery (HADR)
    - overview 39
    - requirements 140
  - overview 356
- disk mirroring 344
- disks
  - failure management 344
  - redundant array of independent disks (RAID) 344
  - striping 344
- DROP STOGROUP statement
  - compatibility with online
    - backups 321
- dual logging 93
- duplexing
  - RAID level 1 344

## E

- errors
  - log full 147
- event monitors
  - High Availability Cluster
    - Multi-Processing (HACMP) for
      - AIX 8
      - IBM PowerHA SystemMirror for
        - AIX 8
- examples
  - alternate server 33

- examples *(continued)*
  - automatic client reroute 33
- export utility
  - online backup compatibility 321

## F

- failback operations 252
- failover
  - AIX 8
  - failover policies 50
  - overview 6
  - performing 242, 247, 248
  - Solaris Operating System 17
  - Windows 13
- fault monitor
  - configuring
    - db2fm command 37
    - db2fmcu command 38
    - system commands 38
  - overview 35, 244
  - registry file 36
- fault tolerance 17

## G

- group crash recovery
  - details 255
  - initiating 265
- group restart
  - details 255
  - overview 254
- guest members
  - details 256

## H

- HADR 352
  - active standby database
    - isolation level 236
    - replay-only window 236
  - automatic client reroute 115
  - cluster managers 133
  - commands 198
  - configuring 118
    - NAT 142
  - data concurrency 235
  - databases
    - activating 177
    - deactivating 177
  - Db2 pureScale environments
    - adding members 225
    - CFs 219
    - member subsetting 219
    - overview 218
    - preferred replay member 223
    - replay member 223
    - restrictions 219
    - scenario 229
    - setup 219
    - standby members 219
    - standby replay 223
    - topology changes 225
  - designing solution 139
  - failback 252

HADR *(continued)*

- failover
  - multiple standbys 207
  - performing 248
  - pureScale environment 227
- initializing
  - multiple standbys 110
  - pureScale environment 110
  - single standby 110
- load operations 118
- log archiving 129
- log flushes 235
- managing 197
- monitoring
  - Db2 pureScale environment 220
  - methods 245
  - multiple standby databases 205
- multiple standbys 200
- multiple standbysrestrictions 200
- non-replicated operations 187
- overview 39
- performance 130
- primary reintegration 252
- quiescing table spaces 194
- replicated operations 186
- requirements 139, 140
- restrictions 143
- rolling updates 179, 182, 204
- rolling upgrades
  - multiple standby databases 204
- standby databases
  - determining state 192
  - initializing 109
  - log spooling 128
  - recovering from table space
    - errors 193
  - states 188
  - synchronizing with primary
    - database 185
- stopping 176
- switching database roles 251
- synchronization modes
  - ASYNCR 134
  - effective 134, 202
  - NEARSYNCR 134
  - operational 134, 202
  - SUPERASYNCR 134
  - SYNCR 134
- takeover
  - multiple standbys 207
  - pureScale environment 227
- HADR multiple standbys
  - adding auxiliary standbys 201
  - changing the principal standby 201
  - configuring 208
  - enabling 110
  - example 208
  - modifying your setup 201
  - monitoring 205
  - NAT support 142
  - overview 200
  - setting up 208
  - takeover
    - examples 214
- HADR reads on standby
  - enabling 235
  - overview 234

- HADR reads on standby *(continued)*
  - restrictions 241
  - terminating read applications 240
- hadr\_peer\_window database
  - configuration parameter
    - automatic reconfiguration 202
    - high availability disaster recovery (HADR) 118
    - setting parameter 127
- hadr\_remote\_host configuration
  - parameter
    - automatic reconfiguration 202
- hadr\_remote\_inst configuration
  - parameter
    - automatic reconfiguration 202
- hadr\_remote\_svc configuration parameter
  - automatic reconfiguration 202
- hadr\_replay\_delay database configuration
  - parameter
    - HADR delayed replay 194
- hadr\_syncmode configuration parameter
  - automatic reconfiguration 202
- hadr\_syncmode database configuration
  - parameter
    - high availability disaster recovery (HADR) 118
- hadr\_timeout configuration parameter
  - setting parameter 127
- hadr\_timeout database configuration
  - parameter
    - high availability disaster recovery (HADR) 118
- hardware
  - disk arrays 344
- heartbeats
  - High Availability Cluster
    - Multi-Processing (HACMP) for AIX 8
  - IBM PowerHA SystemMirror for AIX 8
  - monitoring 242, 244
  - Solaris 17
- high availability
  - administering 163
  - configuring
    - AUTO\_DEL\_REC\_OBJ
      - parameter 294
    - clustered environments 161
    - overview 106
  - Db2 server features 26
  - designing 1, 139
  - fault monitor
    - configuring (db2fm command) 37
    - configuring (db2fmcu and system commands) 38
    - overview 244
    - registry file 36
  - heartbeat monitoring 244
  - keepalive timeout 107
  - non-JDBC 108
  - log shipping 91
  - maintenance
    - minimizing impact 175
  - Microsoft Cluster Server (MSCS) 13
  - outages
    - avoiding 5
    - cost 4

- high availability *(continued)*
  - outages *(continued)*
    - detecting 242, 244
    - overview 1, 2
    - responding 242, 247
    - signatures 2
    - tolerance 4
  - Solaris Operating System 17
  - strategies
    - clustering 7
    - failover 6
    - overview 5
    - redundancy 6, 185
  - Tivoli System Automation for Multiplatforms 11
- High Availability Cluster
  - Multi-Processing (HACMP)
    - see IBM PowerHA SystemMirror for AIX 8
- high availability disaster recovery
  - see HADR 39
- High Availability Disaster Recovery
  - see HADR 39
- high availability disaster recovery (HADR)
  - Db2 pureScale environment
    - preferred replay member 224
    - removing members 226
    - topology changes 226
  - preferred replay member 224
- history file
  - accessing 287
- hot standby configuration
  - overview 8
- HP-UX
  - backups 277
  - restores 277

**I**

- IBM PowerHA SystemMirror for AIX
  - details 8
- IBM Tivoli Storage Manager (TSM)
  - data recovery 436
- IBM Tivoli System Automation for Multiplatforms (SA MP)
  - overview 43
- images
  - backing up 300
- incremental backups
  - details 361
  - images for rebuilding databases 398
- incremental recovery
  - overview 361
- incremental restores
  - overview 374
  - restoring from incremental backup
    - images 363
- indexes
  - logging for high availability disaster recovery (HADR) 116
- indoubt transactions
  - recovering
    - with Db2 syncpoint manager 350
    - without Db2 syncpoint manager 351

- initializing HADR
  - multiple standbys 110
  - pureScale environment 110
  - single standby 110
- instance\_name.nfy log file 243

## K

- keepalive packets 8

## L

- Linux
  - backup and restore operations
    - between different operating systems and hardware platforms 277
- local catchup state 188
- log mirroring
  - details 93
  - synchronizing databases 185
- log record identifiers (LRIs)
  - Db2 pureScale environments 283
- log replay error
  - resolving 185
- log sequence numbers (LSNs)
  - Db2 pureScale environments 283
- log shipping
  - details 91
  - synchronizing database servers 185
- log spooling
  - overview 128
- log stream merges
  - overview 279
- log streams
  - overview 279
- logarchmeth1 configuration parameter
  - high availability disaster recovery (HADR) 129
- logarchmeth2 configuration parameter
  - high availability disaster recovery (HADR) 129
- logarchopt1 configuration parameter
  - cross-node recovery examples 326
- logbufsz database configuration
  - parameter
    - overview 147
- logfilsiz database configuration parameter
  - high availability disaster recovery (HADR) 118
  - overview 147
- logging
  - reducing 156
- logprimary database configuration
  - parameter
    - overview 147
- logs
  - active 23
  - administering 243
  - allocating 171
  - archive logging 24, 171
  - archived
    - compression 275, 276
  - circular logging 23, 171
  - configuring 146
  - control files 25

- logs (*continued*)
  - databases
    - overview 23
  - Db2 pureScale environments 279
  - directory 157
  - including in backup image 173
  - indexes 116
  - log archiving 129, 158, 165
  - log control files 25
  - loss prevention 175
  - managing
    - overview 163
  - offline archived 24
  - online archived 24
  - removing 171
  - space requirements
    - recovery 274
  - user exit programs 274
- logsecond configuration parameter
  - overview 147
- LRIs (log record identifiers)
  - Db2 pureScale environments 283
- LSNs (log sequence numbers)
  - Db2 pureScale environments 283

**M**

- maintenance
  - scheduling 143
- media failures
  - catalog partitions 344
  - logs 274
  - reducing impact 344
- member crash recovery
  - details 254
  - initiating 266
- member restart
  - details 254
  - overview 254
- members
  - crash recovery
    - details 254
    - initiating 266
  - resident 256
  - restarting
    - details 254
    - overview 254
- Microsoft Failover Clustering server 13
- mincommit database configuration
  - parameter
    - overview 147
- mirrorlogpath database configuration
  - parameter
    - overview 93, 147
- MON\_GET\_HADR table function
  - HADR standby database states 192
- monitoring
  - backups 318
  - high availability disaster recovery (HADR)
    - Db2 pureScale environment 220
    - multiple standby databases 205
    - overview 245
  - restores 409, 430
- multiple instances
  - Tivoli Storage Manager 439
- mutual takeover configuration 8

**N**

- named pipes
  - backing up to 309
- NEARSYNC synchronization mode 134
- newlogpath database configuration
  - parameter
    - overview 147
- nodedown event 8
- nodes
  - synchronization 161
- nodeup event 8
- nonrecoverable databases
  - backup and recovery strategy 269

**O**

- offline archived logs 24
- offline backups
  - compatibility with online
    - backups 321
- offline loads
  - compatibility with online
    - backups 321
- on demand log archiving 165
- online archived logs 24
- online backups
  - compatibility with other utilities 321
- online index creation
  - compatibility with online
    - backups 321
- online index reorganization
  - compatibility with online
    - backups 321
- online inspect
  - compatibility with online
    - backups 321
- online loads
  - compatibility with online
    - backups 321
- online table reorganization
  - compatibility with online
    - backups 321
- optimization
  - backup performance 319
  - restore performance 410
- overflowlogpath database configuration
  - parameter
    - overview 147

**P**

- parallelism
  - recovery 366
- partitioned database environments
  - backing up 310
  - rebuilding databases 399
  - transactions
    - failure recovery 346
- partitioned tables
  - backing up 311
- peer states
  - details 188
- pending states 188
- performance
  - high availability disaster recovery (HADR) 130

- performance (*continued*)
  - recovery 366
- points of consistency
  - database 341
- primary cluster caching facilities
  - automated failover 254
- primary database connections
  - configuration parameters 127
- primary database reintegration after takeover 252
- privileges
  - backup utility 321
  - restore utility 410
  - rollforward utility 432
- proxy nodes
  - Tivoli Storage Manager (TSM)
    - configuration 437
    - example 326

**Q**

- quiescing
  - HADR environment 194
- quorum devices 49

**R**

- RAID devices
  - data striping 344
  - disk mirroring 344
  - duplexing 344
  - level 1 344
  - level 5 344
  - parity striping 344
  - reducing impact of media failure 344
- rebalancing
  - compatibility with online
    - backups 321
  - table spaces 178
- RECOVER DATABASE command
  - authorities required 367
  - privileges required 367
  - recovering data 325
- recoverable databases
  - details 269
- recovery
  - after failure of database partition
    - server 350
  - crash 341
  - cross-node examples 326
  - damaged table spaces 266, 342, 343
  - databases
    - overview 324
    - rebuilding 389
  - dropped tables 339
  - history file 283
  - incremental 361
  - operating system restrictions 277
  - parallel 366
  - performance 366
  - point-in-time 358
  - reducing logging 156
  - rollforward 358
  - storage considerations 274
  - strategy overview 269
  - time required 272

- recovery (*continued*)
    - Tivoli Storage Manager (TSM) proxy
      - nodes example 326
      - to end of logs 358
      - two-phase commit protocol 346
      - version 357
  - recovery history file
    - active entry status 285
    - do\_not\_delete entry status 285, 290
    - entries
      - protecting 290
      - pruning 288
    - expired entry status 285
    - inactive entry status 285
    - pruning
      - automated 289
      - causes 294
      - db2Prune API 288
      - PRUNE HISTORY command 288
  - recovery objects
    - deleting
      - automating 292
      - db2Prune API 292
      - methods 291
      - PRUNE HISTORY command 292
    - protecting from being deleted 294
  - redirected restore
    - table space 185
  - redirected restores
    - overview 381
    - using generated script 385, 387
  - redundancy 6
  - registry variables
    - DB2\_HADR\_PEER\_WAIT\_LIMIT 130
    - DB2\_HADR\_SORCVBUF 130
    - DB2\_HADR\_SOSNDBUF 130
  - remote catchup pending state 188
  - remote catchup state 188
  - RENAME STOGROUP statement
    - compatibility with online
      - backups 321
  - reorganization
    - tables
      - compatibility with online
        - backups 321
  - replay delay
    - HADR configuration 194
    - HADR standby 194, 195
  - replicated operations
    - high availability disaster recovery (HADR) 186, 187
  - resident members
    - details 256
  - resource groups 48
  - resources
    - overview 48
  - RESTART DATABASE command
    - crash recovery 341
  - restart light
    - disabling automatic failback 257
    - example 262
    - memory usage 258
    - monitoring 261
    - overview 256
  - restore Db2 Enterprise Server Edition to Db2 pureScale Feature 418
  - restore Db2 pureScale Feature to Db2 Enterprise Server Edition 417
  - restore utility
    - authorities required 410
    - compatibility with online
      - backups 321
    - Db2 pureScale environments 313
    - examples 381
    - monitoring progress 409, 430
    - overview 367
    - performance 367, 410
    - privileges required 410
    - redefining table space containers 381
    - redirected restores 381
    - restoring data 368
    - restoring to existing database 373
    - restoring to new database 374
    - restrictions 368
    - topology change 376
  - restores
    - automatic incremental 364
    - from snapshot backup 371
    - incremental 361, 363, 374
    - rollforward recovery 358
    - snapshot backup
      - with script 372
    - statistics 320
    - to existing database 373
    - to new database 374
    - transporting database schemas
      - examples 414
      - overview 411
      - transportable objects 413
      - troubleshooting 417
  - return codes
    - user exit programs 170
  - ROLLFORWARD DATABASE command
    - applying transactions to restored
      - backup image 421
    - Db2 pureScale environments 427
  - rollforward recovery
    - configuration parameters 147
    - databases 358
    - log management 163
    - minimum recovery time 423
    - table spaces 358, 423
    - through a topology change 430
  - rollforward utility
    - authorities required 432
    - compatibility with online
      - backups 321
    - examples 432
    - overview 419
    - privileges required 432
    - recovering dropped table 339
    - recovery from failures 423
    - restarting 423
    - restrictions 421
  - rolling updates
    - performing
      - HADR environments 179, 182
      - multiple standby databases 204
  - rolling upgrades
    - performing
      - multiple standby databases 204
  - rotating assignments 8
  - roving high availability (HA) failover
    - disabling 51
    - enabling 51
  - rstrt\_light\_mem database manager
    - configuration parameter
      - setting 258
  - RUNSTATS utility
    - compatibility with online
      - backups 321
- S**
- samples
    - automatic maintenance 145
  - scalability
    - multi-clustered databases 8
  - secondary cluster caching facilities
    - automated failover 254
  - server clustering 13
  - servers
    - alternate 26, 30
  - SET WRITE command
    - compatibility with online
      - backups 321
  - site failures
    - high availability disaster recovery (HADR) 39
  - snapshot backups
    - activating Db2 Advanced Copy Services (ACS) 443
    - managing snapshot backup
      - objects 294
    - performing 305
      - with script 306
    - restoring
      - with script 372
    - restoring from 371
  - software disk arrays 344
  - Solaris operating systems
    - backups 277
    - restores 277
  - split mirrors
    - backup images
      - Db2 pureScale environment 105
      - procedure 104
    - clone databases
      - Db2 pureScale environment 101
      - procedure 100
    - overview 93
    - standby databases
      - Db2 pureScale environment 96
      - outside a Db2 pureScale environment 94
  - START HADR command
    - starting HADR 198
  - states
    - standby database 188
  - STOP HADR command
    - overview 198
  - storage
    - media failures 274
    - requirements
      - backup 274
      - recovery 274
  - SUPERASYNC synchronization
    - mode 134



- suspended I/O
  - disk mirroring 185
  - overview 93
- switching
  - database roles 251, 252
- sync point manager (SPM)
  - recovering indoubt transactions 350
- SYNC synchronization mode 134
- synchronization
  - HADR synchronization mode 134
  - partitioned database environments 161
- system clock
  - changing 184
- system requirements
  - high availability disaster recovery (HADR) 139

## T

- table space containers
  - redefining by restoring database by using script 385
  - redefining in redirected restore operation 381
- table spaces
  - containers
    - rebuilding databases 393
  - damaged 266
  - rebalancing 178
  - rebuilding 389, 397
  - recovering
    - damaged table spaces 266, 342
    - non-recoverable databases 343
    - recoverable databases 343
    - rollforward recovery 423
    - rollforward recovery 358
  - rollforward recovery
    - details 358, 423
- tables
  - recovering dropped tables 339
  - related data 277
- TAKEOVER HADR command
  - overview 198
  - performing failover operations 248
  - switching database roles 251
- tape backups
  - procedure 307
- tape drives
  - storing log files on 158, 165
- target images
  - database rebuilds 394
- TCP\_KEEPALIVE operating system
  - configuration parameter 29
- TCP/IP
  - configuring
    - high availability 107, 108
- temporary table spaces
  - database rebuilds 393
- time change 184
- time stamps
  - conversion in client/server environment 162
- Tivoli Storage FlashCopy Manager
  - installing 446
  - restrictions 441
  - setup script setup\_db2.sh 445

- Tivoli Storage Manager
  - client configuration 437
  - dsmapiw command 437
  - partitioned tables 311
  - recovery example 326
  - server configuration 439
  - upload examples 295
- Tivoli System Automation for Multiplatforms (Tivoli SA MP)
  - AIX 11
  - Linux 11
- transactions
  - blocking when log directory is full 157
  - failures
    - recovery in partitioned database environment 346
    - reducing impact 341, 346
- transports of database schemas
  - examples 414
  - overview 411
  - transportable objects 413
  - troubleshooting 417
- TRUNCATE statement
  - compatibility with online backups 321
- two-phase commit
  - partitioned database environments 346

## U

- unplanned outages
  - detecting 244
- user exit programs
  - archiving log files 158
  - calling format 169
  - database recovery 167
  - error handling 170
  - retrieving log files 158
  - sample programs
    - UNIX 168
    - Windows 168
- user-defined events 8

## V

- vendoropt configuration parameter
  - cross-node recovery examples 326
- VERITAS Cluster Server 19
- version recovery of databases 357

## W

- Windows
  - failover 13







Printed in USA