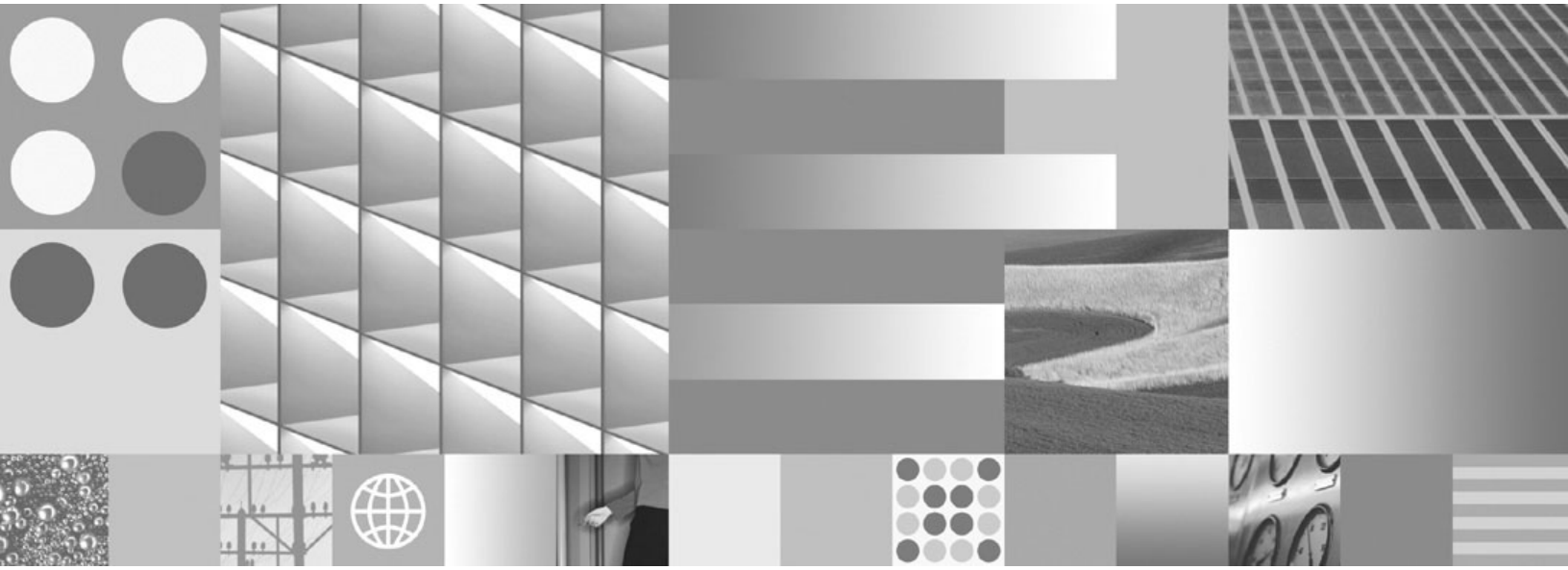
IBM Informix

Version 11.50

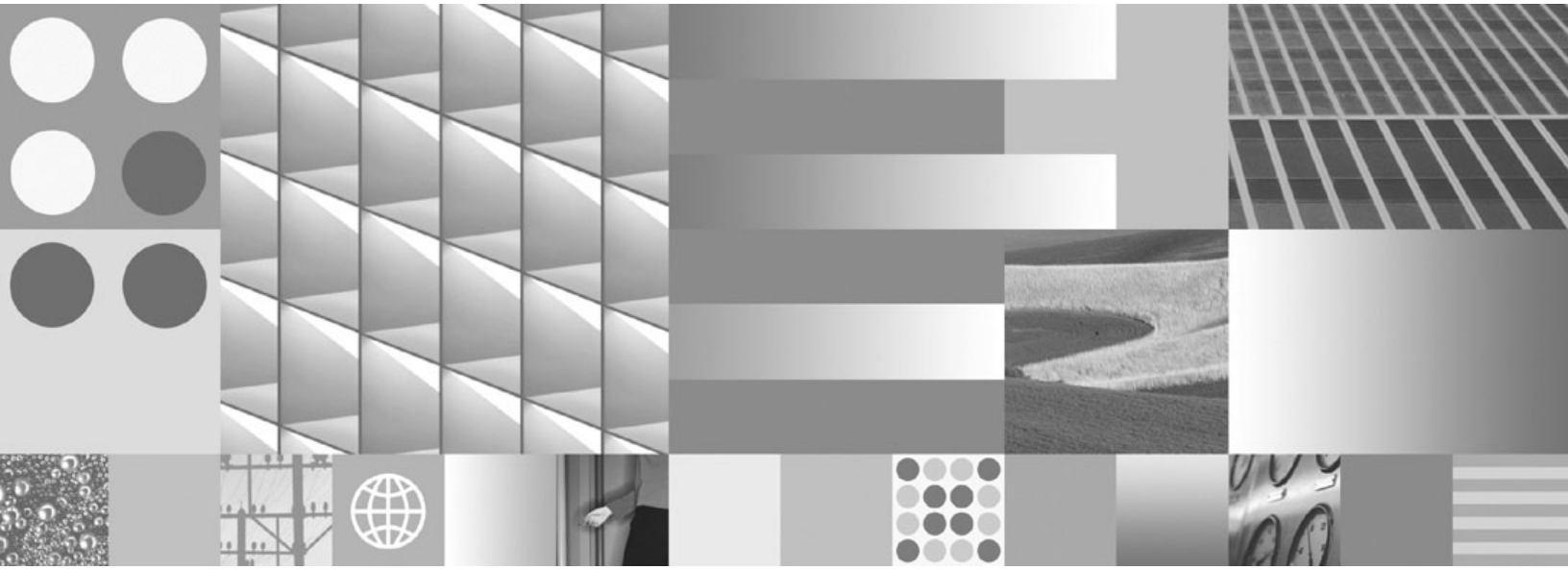


IBM Informix Dynamic Server Administrator's Guide

SC23-7748-04

IBM Informix

Version 11.50



IBM Informix Dynamic Server Administrator's Guide

SC23-7748-04

Note
Before using this information and the product it supports, read the information in "Notices" on page B-1.

This edition replaces SC23-7748-03.

Contents

Part 2. Disk, Memory, and Process Management

Part 3. Logging and Log Administration

Introduction

In This Introduction

This introduction provides an overview of the information in this publication and describes the conventions it uses.

About This Publication

This publication describes concepts and procedures for configuring, administering, and using IBM® Informix® Dynamic Server (IDS) or IBM Informix Dynamic Server with J/Foundation.

A companion volume, the *IBM Informix Dynamic Server Administrator's Reference*, contains reference material for using Informix database servers. If you need to tune the performance of your database server and SQL queries, see your *IBM Informix Dynamic Server Performance Guide*.

Types of Users

This publication is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Performance engineers
- Programmers in the following categories
  - Application developers
  - DataBlade® module developers
  - Authors of user-defined routines

This publication is written with the assumption that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
- Some experience with database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, see the *IBM Informix Dynamic Server Getting Started Guide* for your database server for a list of supplementary titles.

Software Dependencies

This publication is written with the assumption that you are using IBM Informix Dynamic Server (IDS) Version 11.50 as your database server.

Assumptions About Your Locale

IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation, and representation of numeric data, currency, date, and time is brought together in a single environment, called a Global Language Support (GLS) locale.

The examples in this publication are written with the assumption that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for date, time, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *IBM Informix GLS User's Guide*.

Demonstration Database

The DB–Access utility, which is provided with your Informix database server products, includes one or more of the following demonstration databases:

- The **stores_demo** database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. Many examples in IBM Informix manuals are based on the **stores_demo** database.
- The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines.

For information about how to create and populate the demonstration databases, see the *IBM Informix DB–Access User's Guide*. For descriptions of the databases and their contents, see the *IBM Informix Guide to SQL: Reference*.

The scripts that you use to install the demonstration databases reside in the **$INFORMIXDIR/bin** directory on UNIX® and in the **%INFORMIXDIR%\bin** directory on Windows®.

What's New in Administration for Dynamic Server, Version 11.50

This publication includes information about new features and changes in existing functionality.

The following changes and enhancements are relevant to this publication. For a comprehensive list of all new features for this release, see the *IBM Informix Dynamic Server Getting Started Guide*.

+ *Table 1. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC5*

+ **Overview**	**Reference**
+ Configuring RS Secondary Server Latency for Disaster Recovery + + To aid in disaster recovery scenarios, you can now configure RS secondary servers to wait for a specified period of time before applying logs. Delaying the application of log files allows you to recover quickly from erroneous database modifications by restoring from the RS secondary server.	"Delayed Application of Log Records" on page 21-7 This feature is documented in the *IBM Informix Dynamic Server Administrator's Guide* and *IBM Informix Dynamic Server Administrator's Reference*.
+ Forcing the Database Server to Shut Down + + Use the onclean utility to force the database server to shut down when normal shut down with the onmode utility fails to shut down the server or when you cannot restart the server. The onclean utility cleans up shared memory, semaphores, and stops database server virtual processes. + Use the onshutdown script to try to shut down the server normally. If the server does not shut down after a specified time, the server is forced to shut down.	"Deleting Shared Memory Segments After a Server Failure" on page 8-11 *IBM Informix Dynamic Server Administrator's Reference*

+

Table 2. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC4

Overview	Reference
Save Disk Space by Compressing Data You can now use SQL administration API commands to save disk space by compressing row data in a table or in one or more table fragments. You can also use SQL administration API commands to consolidate free space in a table or fragment, return this free space to the dbspace, and estimate the amount of space that is saved by compressing the data. You can display the following types of information about compression: • Active compression dictionaries that describe how the data is compressed, with the new **onstat -g ppd** command • All compression dictionaries, by querying the new **syscompdicts_full** table and **syscompdicts** view in the **sysmaster** database • Progress of currently running compression operations, with the new **onstat-g dsk** command • Uncompressed contents of compressed log records, with a new **onlog** utility option • Percentage of compressed rows, with the **onstat -pT** option Before you can compress a table or fragment, you must run an SQL administration API command that enables compression. If you enable compression on the version 11.50.xC4 server and you want to revert to an earlier version of the server, which does not support compression, you must uncompress or drop any compressed table or fragments before you revert. To ensure successful reversion, follow the Dynamic Server reversion procedures that are described in the Migration Guide. You can use the **onmode -b** command to revert to Version 11.50.xC1, 11.50.xC2, or 11.50.xC3 if you previously used that version of the server. If you are migrating to Version 11.50.xC4 from Version 11.50.xC1, 11.50.xC3, or 11.50.xC3, you must run the **buildsmi** script if you plan to use the **syscompdict_full** table. The **buildsmi** script, which is in the **etc** directory in your installation, drops and recreates the **sysmaster** database, which contains the **syscompdict_full** table. The **buildsmi** script must be run as user **informix** on UNIX, or as a member of the **Informix-Admin** group on Windows, after ensuring that no connections to the sysmaster database are made during the build of the database.	"Compression of Row Data and Storage Optimization" on page 10-46 *IBM Informix Dynamic Server Administrator's Reference*
Disable IPv6 Support You can now optionally disable Internet Protocol Version 6 (IPv6) support for better performance on certain platforms. You can disable IPv6 support for one or more database instances or client applications, or for all instances and client applications.	This information is documented in the *IBM Informix Dynamic Server Administrator's Guide*.

Table 2. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC4 (continued)

Overview	Reference
ONINIT_STDOUT environment variable allows capturing oninit output (Windows) You can now capture the output of the **oninit** command on Windows systems by setting the new ONINIT_STDOUT environment variable. Capturing the output of the **oninit** command is useful in certain situations such as when using the **-v** (verbose) option or when you want to see output from an unhandled exception in a process launched within a virtual processor.	This feature is documented in the *IBM Informix Dynamic Server Administrator's Guide* and the *IBM Informix Guide to SQL: Reference*.
Changing how IDS Searches for User Names (Windows) You can now use the CHECKALLDOMAINSFORUSER configuration parameter to configure how IDS searches for user names logging in to a networked Windows environment. You set the CHECKALLDOMAINSFORUSER to determine whether IDS searches for the user name on the local host only or on all domains.	This information is documented in the *IBM Informix Dynamic Server Administrator's Guide* and *IBM Informix Dynamic Server Administrator's Reference*.

Table 3. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC3

Overview	Reference
Changing the Size of the First Extent of a Table When you create a table, you specify a first extent size based on the eventual estimated size of the table. If the table becomes larger or smaller than that estimate, you might want to change the size of the first extent to avoid either having too many extents or creating extents that are larger than necessary. You can now change the size of the first extent of a table in a dbspace. When you change the size of the first extent, Dynamic Server records the change (in the system catalog and on the partition page), but only makes the actual change when the table is rebuilt or a new partition or fragment is created.	*IBM Informix Guide to SQL: Syntax*

Table 4. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC2

Overview	Reference
Reconfiguring Connection Manager while It's Running In past releases, you had to stop the Connection Manager to modify how it was configured, and then restart it for the new configuration to take effect. With the new reload option (-r), you can reconfigure the Connection Manager without stopping and restarting it.	"Modifying Connection Manager Options" on page 23-8

Table 4. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC2 (continued)

Overview	Reference
Limiting the Number of Sessions That Can Connect to Dynamic Server You can now limit the number of sessions that can connect to the server. You do this by setting the LIMITNUMSESSIONS configuration parameter to the maximum number of sessions that you want connected to the database server. Optionally, you can also specify whether you want the server to print messages when the number of sessions approaches a specified maximum number. You can use **onmode -wm** and **onmode -wf** commands to turn this configuration parameter on or off or change the value of the configuration parameter.	"LIMITNUMSESSIONS Configuration Parameter" on page 3-34

Table 5. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC1

Overview	Reference
Enhanced Connection Management for High-Availability Clusters The new Connection Manager dynamically routes client application connection requests to the most appropriate server in a high-availability cluster. Connection Manager connects to each of the servers in the cluster and gathers statistics about the type of server, unused workload capacity, and the current state of the server. From this information, the Connection Manager redirects the connection to the appropriate server. In addition, Connection Manager Arbitrator provides automatic failover logic for high-availability clusters. Using a configuration file, you specify which secondary server takes over if the primary server fails.	Chapter 23, "Manage Cluster Connections with the Connection Manager," on page 23-1
Enhanced troubleshooting for high-availability cluster environments New onstat and sysmaster options are available for easier configuration and troubleshooting of high-availability cluster environments.	"Troubleshooting High-Availability Cluster Environments" on page 19-26
Update Data Support on High-Availability Cluster Secondary Servers Client applications can now update data on secondary servers by using *distributed writes* (sometimes called *redirected writes*). Distributed writes give the appearance that updates are occurring directly on the secondary server; however, the transactions are transferred to the primary server and then the changes are propagated back to the secondary server. Distributed writes are not enabled by default. To enable distributed writes, set the REDIRECTED_WRITES configuration parameter to the number of SMX pipes between the primary and secondary servers.	"Update Data on Secondary Servers" on page 19-11

Table 5. What's New in IBM Informix Dynamic Server Administrator's Guide for Version 11.50.xC1 (continued)

Overview	Reference
Determine Data Currency with a Version Column Use row versioning to determine whether a row was changed and to detect collisions. With row versioning enabled, each row of a table is configured to contain both a checksum and a version number. If a row is deleted and another row is re-inserted in a table, it is possible to recognize that the row is different. By comparing the row checksum and row version between the secondary and the primary servers, it is possible to detect data collisions.	"Row Versioning" on page 19-14
DRDA® enhancements, including: • Support for the Secure Sockets Layer (SSL) protocol for DRDA communications • The new **syssesappinfo** table in the **sysmaster** database for viewing Data Server client session information • Support for retrieving ISAM errors, connecting to a database that has a name with more than 18 characters, and the batching of multiple statements into one request	"Distributed Relational Database Architecture (DRDA) Communications" on page 3-36
Monitor storage spaces and partitions You can now set the threshold level and time interval for alarms that are raised when storage spaces or partitions become full.	"Monitoring Storage Spaces" on page 10-7
Create transient data types on secondary servers Transient unnamed complex data types (ROW, SET, LIST, and MULTISET) can now be used on high-availability cluster secondary servers, whether the secondary servers are read-only or use distributed writes.	"Transient Types on High-Availability Cluster Secondary Servers" on page 19-14
Create secondary server alias names You specify the name of an RS secondary or SD secondary server to a high-availability cluster configuration using the HA_ALIAS configuration parameter. Connection Manager Arbitrator can use this alias name to fail over to a secondary server.	"HA_ALIAS Configuration Parameter" on page 3-35
Specify a failover script You can specify the name of a script to run when a database server transitions from a secondary server to a primary server, or from a secondary server to a standard server using the FAILOVER_CALLBACK configuration parameter.	"Failover with ISV Cluster Management Software" on page 23-8

Documentation Conventions

Special conventions are used in the product documentation for IBM Informix Dynamic Server.

Technical Changes

Technical changes to the text are indicated by special characters depending on the format of the documentation.

HTML documentation

New or changed information is surrounded by blue ⩾ and ⩽ characters.

PDF documentation

A plus sign (+) is shown to the left of the current changes. A vertical bar (|) is shown to the left of changes made in earlier shipments.

Feature, Product, and Platform Markup

Feature, product, and platform markup identifies paragraphs that contain feature-specific, product-specific, or platform-specific information.

Some examples of this markup follow:

Dynamic Server only: Identifies information that is specific to IBM Informix Dynamic Server

Windows only: Identifies information that is specific to the Windows operating system

This markup can apply to one or more paragraphs within a section. When an entire section applies to a particular product or platform, this is noted as part of the heading text, for example:

Table Sorting (Windows)

Example Code Conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement. If you are using DB–Access, you must delimit multiple statements with semicolons.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the documentation for your product.

Additional Documentation

Documentation about IBM Informix products is available in various formats.

You can view, search, and print all of the product documentation from the IBM Informix Dynamic Server information center on the Web at http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp.

For additional documentation about IBM Informix Dynamic Server and related products, including release notes, machine notes, and documentation notes, go to the online product library page at http://www.ibm.com/software/data/informix/pubs/library/. Alternatively, you can access or install the product documentation from the Quick Start CD that is shipped with the product.

Compliance with Industry Standards

IBM Informix products are compliant with various standards.

IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

The IBM Informix Geodetic DataBlade Module supports a subset of the data types from the *Spatial Data Transfer Standard (SDTS)—Federal Information Processing Standard 173*, as referenced by the document *Content Standard for Geospatial Metadata*, Federal Geographic Data Committee, June 8, 1994 (FGDC Metadata Standard).

IBM Informix Dynamic Server (IDS) Enterprise Edition, Version 11.50 is certified under the Common Criteria. For more information, refer to *Common Criteria Certification: Requirements for IBM Informix Dynamic Server*, which is available at http://www.ibm.com/support/docview.wss?uid=swg27015363.

How to Provide Documentation Feedback

You are encouraged to send your comments about IBM Informix user documentation.

Use one of the following methods:

- Send e-mail to docinf@us.ibm.com.
- Go to the information center at http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp and open the topic that you want to comment on. Click the feedback link at the bottom of the page, fill out the form, and submit your feedback.

Feedback from both methods is monitored by those who maintain the user documentation. The feedback methods are reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact IBM Technical Support. For instructions, see the IBM Informix Technical Support Web site at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Part 1. The Database Server

Chapter 1. Installing and Configuring the Database Server

In This Chapter

When you install IBM Informix Dynamic Server (IDS), you should follow the installation instructions to ensure the permissions of all key files and directories are set appropriately. The install instructions are in the *IBM Informix Dynamic Server Installation Guide for UNIX, Linux, and Mac OS X* and the *IBM Informix Dynamic Server Installation Guide for Windows*.

After you install a new version of the database server, you must configure it.

Configuration refers to setting specific parameters that customize the database server for your data-processing environment: quantity of data, number of tables, types of data, hardware, number of users, and security needs.

The following sections of this chapter contain information on basic configuration and server startup tasks:

This chapter also contains information on downloading the OpenAdmin Tool for IDS, which provides the ability to administer multiple database server instances from a single location. See "The OpenAdmin Tool for IDS" on page 1-27.

When you start a database server on a UNIX or Linux® computer, server utilities check to determine if your environment is secure. For more information, see the *IBM Informix Security Guide*

Plan for the Database Server

Configuring a database management system requires many decisions, such as where to store the data, how to access the data, and how to protect the data. How you install and configure Dynamic Server can greatly affect the performance of database operations.

You can customize the database server so that it functions optimally in your particular data-processing environment. For example, using a database server to serve 1000 users who execute frequent, short transactions is very different from using a database server to serve a few users making long and complicated searches.

When you are planning for your database server, consider your priorities and your environment.

Considering Your Priorities

As you prepare the initial configuration and plan your backup strategy, keep in mind the characteristics of your database server:

- Do applications on other computers use this database server instance?
- What is the maximum number of users that you can expect?
- To what extent do you want to control the environment of the users?
- Are you limited by resources for space, CPU, or availability of operators?
- Do you need the database server to run unsupervised?
- Does the database server usually handle many short transactions or fewer long transactions?
- Which new database server features or related products do you plan to use?

Considering Your Environment

Before you start the initial configuration of your database server, collect as much of the following information as possible. You might need the assistance of your system administrator to obtain this information:

- The host names and IP addresses of the other computers on your network
- Does your UNIX platform support the Network Information Service (NIS)?
- The disk-controller configuration

  How many disk drives are available? Are some of the disk drives faster than others? How many disk controllers are available? What is the disk-controller configuration?
- What are the requirements, features, and limitations of your storage manager and backup devices?

  For more information, see the *IBM Informix Storage Manager Administrator's Guide* or your storage-manager documentation.
- Do you need to upgrade the hardware and operating system? Is your operating system 32-bit or 64-bit?
- Operating-system shared memory and other resources

  How much shared memory is available? How much of it can you use for the database server?

UNIX only: The machine notes file indicates which parameters are applicable for each UNIX platform.

Configure the Operating System

Before you can start configuring the database server, you must configure the operating system appropriately. You might need the assistance of the system administrator for this task.

The 32-bit version of Dynamic Server runs on a 64-bit or 32-bit operating system. The 64-bit version of Dynamic Server must run on a 64-bit operating system. For more information, see "Memory Use on 64-Bit Platforms" on page 7-31.

Configuring Windows Memory

On Windows, you must create NTFS partitions and configure memory. Also see "Providing NTFS Partitions in Windows" on page 1-5.

Insufficient memory for the database server can result in excessive buffer-management activity. When you set the Virtual Memory values in the System application on the Control Panel, ensure that you have enough paging space for the total amount of physical memory.

Maximum Address Space

On Windows, the maximum address space per computer system is:

- 1.7 gigabytes if the **boot.ini** file is not modified to 3 gigabytes.
- 2.7 gigabytes if the **boot.ini** file is modified to 3 gigabytes.

You can use the **/userva=xxxx** switch for more precise tuning of user and kernel virtual memory space. Use this switch with the 3-gigabyte switch in the **boot.ini** file to tune the user-mode space to a value that is between 2 and 3 gigabytes, with the difference (3,072 less xxxx) given back to the kernel mode. Note that xxxx is a value expressed in megabytes.

The following sample **boot.ini** file shows how to use the new switch to tune a computer to allocate 2,900 megabytes of user-mode virtual memory and 1,196 megabytes of kernel-mode virtual memory. This increases the available kernel space by 172 megabytes.

```
[Boot Loader]
Timeout=30
Default=multi(0)disk(0)rdisk(0)partition(2)\WINNT
[Operating Systems]
multi(0)disk(0)rdisk(0)partition(2)\WINNT="Microsoft Windows Server 2003"
/fastdetect /3GB /Userva=2900
```

Modifying UNIX Kernel Parameters

The machine notes file contains recommended values for configuring operating-system resources. Use these recommended values when you configure the operating system.

If the recommended values for the database server differ significantly from your current environment, consider modifying your operating-system configuration. For more information, see your *IBM Informix Dynamic Server Performance Guide*.

On some operating systems, you can specify the amount of shared memory allocated to the database server. The amount of available memory influences the values that you can choose for the shared-memory parameters in your configuration file. In general, increasing the space available for shared memory enhances performance. You might also need to specify the number of locks and semaphores.

For background information on the role of the UNIX kernel parameters, see Chapter 8, "Managing Shared Memory," on page 8-1.

Configure Disk Space

Configure your disks to obtain optimum performance with data marts and data warehouses. Disk I/O is the longest portion of the response time for an SQL operation. The database server offers parallel access to multiple disks on a computer. Before you allocate the disk space, study the information about disk space in your operating-system administration guide and see "Disk Allocation for Chunks" on page 9-2.

Using Large Chunks

The size of chunks for dbspaces is 4 terabytes for a 2-kilobyte page. Chunks can reside anywhere in a 64-bit address space.

To activate the creation of large chunks, you must use the **onmode** utility. The **onmode** utility uses a **-BC** flag to control the availability of large chunks, chunks greater than 2 gigabytes.

When the database server is first migrated to a new version of Dynamic Server, large chunks are not allowed. Run **onmode -BC 1** to allow large chunks to be created.

If necessary to convert older versions of Dynamic Server to the large chunk format, execute **onmode -BC 2.**

If the root chunk is a large chunk at the time the server starts, the **onmode -BC 1** step is skipped and **-BC 2** mode starts automatically.

Creating Chunk Files on UNIX

On UNIX, you can store data in chunks that use either unbuffered (*raw*) disks or operating-system files, also known as *buffered* or *cooked* files.

Raw or Unbuffered Disk Access

UNIX provides unbuffered disk access using character-special devices (also known as *raw* disk devices). To create raw disk devices on UNIX, follow the instructions provided with your operating system.

The database server uses *raw* disk access to improve the speed and reliability of disk I/O operations. Raw disk access bypasses the file-buffering mechanism that the operating system provides. The database server itself manages the data transfers between disk and memory. The database server optimizes table access by guaranteeing that rows are stored contiguously.

Important: While you should use raw disk devices on UNIX to achieve better performance, recent advances in I/O caching for cooked writes can provide similar if not better performance. To determine the best device performance, perform benchmark testing on the system with both types of devices for the dbspace and table layout.

To allocate disks for the database server

1. Configure a raw disk device for each disk.
2. Create standard device names or filenames.
3. Set permissions, ownership, and group for each raw disk device.

Cooked Files

If optimum performance is unimportant, you can configure the database server to store data in *cooked* files. Cooked files are easier to set up than raw disk devices.

Providing NTFS Partitions in Windows

On Windows, install the database server on a New Technology File System (NTFS) or File Allocation Table (FAT) partition. However, you must store all dbspaces, blobspaces, and sbspaces in NTFS files or on a physical drive or logical disk partition. You should use NTFS files to simplify disk administration. For more information about NTFS files, see your Windows documentation and "Disk Access on Windows" on page 9-2.

If all of your partitions are FAT partitions, you need to convert at least one partition to NTFS. You can use the Windows **convert** utility, as the following example shows:

```
convert /fs:ntfs
```

Setting Permissions, Ownership, and Group

Files or raw disk devices that the database server uses must have the appropriate ownership and permissions.

UNIX Only:

On UNIX, the owner and group must be **informix**, and the permissions must be set to read and write for both the user and the group (but not for others).

If you want users other than **informix** or **root** to execute ON–Bar commands, create a **bargroup** group. Only members of **bargroup** can execute ON–Bar commands. The **bargroup** group is not created automatically during database server installation. For instructions on creating a group, see the *IBM Informix Dynamic Server Installation Guide for UNIX, Linux, and Mac OS X* or your UNIX documentation.

Windows Only:

On Windows, files must be owned by a member of the **Informix-Admin** group. The **Informix-Admin** group is created automatically when you install the database server.

Creating Standard Device Names (UNIX)

You should use symbolic links to assign abbreviated standard device names for each raw disk device. If you have symbolic links, you can replace a disk that has failed with a new disk by assigning the symbolic name to the new disk.

To create a link between the character-special device name and another filename, use the UNIX link command (usually **ln)**.

Execute the UNIX command **ls -l** on your device directory to verify that both the devices and the links exist. The following example shows links to raw disk devices. If your operating system does not support symbolic links, you can use hard links.

```
ls -l
crw-rw---  /dev/rxy0h
crw-rw---  /dev/rxy0a
lrwxrwxrwx /dev/my_root@->/dev/rxy0h
lrwxrwxrwx /dev/raw_dev2@->/dev/rxy0a
```

Set Environment Variables

To start, stop, or access a database server, *each user* must hold the required database access privileges, and must set the appropriate environment variables. Some environment variables are required; others are optional.

To set the required environment variables

1. Set **INFORMIXDIR** to the directory where you installed the IBM Informix products.
2. Set the **PATH** environment variable to include **$INFORMIXDIR/bin** (UNIX) or **%INFORMIXDIR%\bin** (Windows).
3. Set **INFORMIXSERVER** to the name of the database server.

Tip: Set the environment variables in the appropriate startup file for your shell file or Windows.

You can include the environment variable "$INFORMIXDIR" in the ONCONFIG file. It should be the first path name value in path name specifications.

The *IBM Informix Guide to SQL: Reference* contains a complete list of environment variables. For information on how setting environment variables can affect performance, see your *IBM Informix Dynamic Server Performance Guide*.

Table 1-1 shows the environment variables that you must set before you access the database server or perform most administrative tasks.

Table 1-1. Required Environment Variables

Environment Variable	Description
CLASSPATH	If you are using J/Foundation, specifies the location of **jvphome/krakatoa.jar** file so that Java™ Development Kit (JDK) can compile the Java source files.
INFORMIXDIR	Specifies the directory where you installed your IBM Informix database server.
INFORMIXSERVER	Specifies the name of the default database server. It has the value specified for the DBSERVERNAME or DBSERVERALIASES configuration parameter.
JVPHOME	If you are using J/Foundation, specifies the directory where you installed the IBM Informix JDBC Driver.
ONCONFIG	Specifies the name of the active **ONCONFIG** file. All users who use database server utilities such as **onstat** must set the **ONCONFIG** environment variable. Users running client applications do not need to set the **ONCONFIG** environment variable. If the **ONCONFIG** environment variable is not present, the database server uses configuration values from the **onconfig** file: On UNIX: **$INFORMIXDIR/etc/onconfig** On Windows: **%INFORMIXDIR%\etc\onconfig.std**
PATH	Specifies the location of executable files. On UNIX: **$INFORMIXDIR/bin** On Windows: **%INFORMIXDIR%\bin**
TERM	Enables DB–Access to recognize and communicate with the terminal that you are using. This environment variable is not required for initialization or start up, but it must be set before you can run an application.

Table 1-1. Required Environment Variables (continued)

Environment Variable	Description
TERMCAP TERMINFO INFORMIXTERM	Specifies whether DB–Access should use the information in the **termcap** file or the **terminfo** directory. If required for your system, you might need assistance from the UNIX system administrator to set these variables because they are highly system dependent.

Setting GLS Environment Variables

The following environment variables let you work with the Global Language Support (GLS) feature. Set these environment variables if you want to use a language other than the default, U.S. English:

- **CLIENT_LOCALE**
- **DB_LOCALE**
- **SERVER_LOCALE**
- **DBLANG**
- **C8BITLEVEL**
- **ESQLMF**
- **GLS8BITFSYS**
- **GL_DATE**
- **GL_DATETIME**

For more information, see the *IBM Informix GLS User's Guide*.

Setting Environment Variables on UNIX

Set UNIX environment variables in one of the following ways:

- At the system prompt on the command line

  When you set an environment variable at the system prompt, you must reassign it the next time you log in to the system.
- In an environment-configuration file such as **$INFORMIXDIR/etc/informix.rc** or **.informix**

  An environment-configuration file is a common or private file where you can set environment variables for each database server user. Use of an environment-configuration file reduces the number of environment variables that you must set at the command line or in a shell file.
- In your **.profile** or **.login** file

  When you set an environment variable in your **.login**, **.cshrc**, or **.profile** file, it is assigned automatically every time you log in to the system. For information about these files, see your operating-system manuals.

To override environment variables that have been automatically set, use a private environment-variable file, **~/.informix**, or assign new values to environment variables individually. For information about the **.informix** and **informix.rc** files, see the *IBM Informix Dynamic Server Administrator's Reference*.

To check the validity of environment variables, use the **chkenv** utility.

Figure 1-1 on page 1-8 shows a sample setup file that contains environment variables for the **miami** database server. **LD_LIBRARY_PATH** is set to the location

of the database server and Informix ESQL/C library files.

```
setenv INFORMIXDIR /ix/informix93
setenv INFORMIXSQLHOSTS /ix/sqlhosts.unified
setenv ONCONFIG s.miami
setenv INFORMIXSERVER miami

# setup to use J/Foundation
setenv JVPHOME /ix/informix93/extend/krakatoa
setenv CLASSPATH $JVPHOME/krakatoa.jar:$JVPHOME/jdbc.jar:/usr/java/lib/classes.zip

# Include jar paths for Java; include /usr/ccs/bin for C compiler:
setenv PATH $INFORMIXDIR/bin:$INFORMIXDIR/extend/krakatoa/krakatoa.jar:
   $INFORMIXDIR/extend/krakatoa/jdbc.jar:/usr/ccs/bin:$PATH

setenv LD_LIBRARY_PATH $INFORMIXDIR/lib:$INFORMIXDIR/lib/esql:/usr/lib
```

Figure 1-1. Sample Setup File

Setting Environment Variables on Windows

On Windows, the installation procedure prepares a file, **setenv.cmd**, that sets the environment variables to their correct values. The **setenv.cmd** file is stored in the **%INFORMIXDIR%** directory. You must execute **setenv.cmd** before you can use any of the command-line utilities.

You can set environment variables or override environment variables that have been automatically set in the following places:

- On Windows, **System > Environment > User Variables**
- In a command-prompt session
- **%INFORMIXDIR%***dbservername***.cmd** batch file

  Use this batch file to configure command-prompt utilities.

For more information, see the *IBM Informix Guide to SQL: Reference* and "Creating an ONCONFIG File on Windows" on page 1-10.

Configure Connectivity

The connectivity information allows a client application to connect to any IBM Informix database server on the network. The connectivity data for a particular database server includes the database server name, the type of connection that a client can use to connect to it, the host name of the computer or node on which the database server runs, and the service name by which it is known.

You must prepare the connectivity information even if the client application and the database server are on the same computer or node.

You do not need to specify all possible network connections in the **sqlhosts** file or registry before you start the database server. But to make a new connection available, you must take the database server offline and then bring it back to online mode once again.

For detailed information about configuring connectivity, see Chapter 3, "Client/Server Communications," on page 3-1.

When you configure connectivity on UNIX, also consider setting the LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTIONS configuration

parameters. These parameters enable you to reduce the risk of a hostile denial-of-service (DOS) attack by making it more difficult to overwhelm the Listener VP that handles connections. For more information, see the *IBM Informix Security Guide*.

The sqlhosts File on UNIX

On UNIX, the **sqlhosts** file contains connectivity information. The default location of this file is **$INFORMIXDIR/etc/sqlhosts**. If you store the information in another location, you must set the **INFORMIXSQLHOSTS** environment variable. For more information, see "The sqlhosts File and the SQLHOSTS Registry Key" on page 3-13.

If you set up several database servers to use distributed queries, use one of the following ways to store the **sqlhosts** information for *all* the databases:

- In one **sqlhosts** file, pointed to by **INFORMIXSQLHOSTS**
- In separate **sqlhosts** files in each database server directory

Use a text editor or ISA to edit the **sqlhosts** file. For more information, see "Configuring Connectivity Using ISA."

Network-Configuration Files

In addition to the **sqlhosts** files, Internet protocol network connections require entries in the **/etc/hosts** and **/etc/services** files. For IPX/SPX connections, the names of the auxiliary files depend on the hardware vendor.

Network-Security Files

IBM Informix database servers follow UNIX security requirements for making connections. Thus, the UNIX system administrator might need to make modifications to the **/etc/passwd**, **etc/hosts**, **~/.rhosts**, and other related files.

The network-configuration and network-security files are described in operating-system manuals.

The sqlhosts Registry on Windows

On Windows, the HKEY_LOCAL_MACHINE registry contains the **sqlhosts** information. The database server installation procedure prepares the registry information. You should not edit the HKEY_LOCAL_MACHINE registry.

Use **setnet32** to manage **sqlhosts** information. For information about **setnet32**, see the installation information in your client documentation. However, **setnet32** does not allow you to assign a database server to a database server group.

Configuring Connectivity Using ISA

Use IBM Informix Server Administrator (ISA) to configure connectivity information for IBM Informix database servers and database server groups for Enterprise Replication. ISA allows you to edit the **sqlhosts** file on UNIX or the **sqlhosts** registry on Windows. For more information, see the ISA onscreen instructions or online help.

In ISA, select **Configuration > SQLHOSTS**.

Configure the Database Server Configuration

The configuration parameters are stored in the **ONCONFIG** file. The product installation script sets the defaults for most of the configuration parameters.

For information about the configuration parameters and how to monitor them, see Chapter 2, "Configuration Parameters," on page 2-1 and the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*. For information on the order of files that the database server checks for configuration values, see "Process Configuration File" on page 4-3.

Preparing the ONCONFIG Configuration File

The **onconfig.std** template file in the **etc** subdirectory of **INFORMIXDIR** contains initial values for many of the configuration parameters. Copy this template and tailor it to your specific configuration.

The template files contain initial values for many of the configuration parameters. You can use a text editor to change the configuration parameters in your **ONCONFIG** file.

Important: Do not modify or delete the template files. The database server provides these files as templates and not as functional configuration files.

Note: If you omit parameters in your copy of the ONCONFIG file, the database server uses values in the **onconfig.std** file for the missing parameters when the server starts.

Tip: The **genoncfg** utility can expedite the process of customizing the configuration file to your hardware and to the anticipated usage of the database server if you do not want to work directly with all the ONCONFIG file parameters. See the *IBM Informix Dynamic Server Administrator's Reference* for more information about this utility.

Creating an ONCONFIG File on UNIX

A new instance of the database server is created and initialized when you install the IBM Informix software for the first time. The installation script automatically creates the **onconfig.demo** file for you.

To prepare the ONCONFIG file using a text editor

1. Copy and rename the **$INFORMIXDIR/etc/onconfig.std** file and store it in the **etc** subdirectory.
2. Use a text editor to edit the **ONCONFIG** configuration file.
3. Set the **ONCONFIG** environment variable to the name of your new **ONCONFIG** file.
4. Initialize the database server if it is a new instance.

   Otherwise, shut down and restart the database server.

Creating an ONCONFIG File on Windows

On Windows, a new instance of the database server is created and initialized when you install the IBM Informix software for the first time. You can also use the

Instance Manager (**instmgr.exe**) to create a new database server instance. The Instance Manager automatically creates an **ONCONFIG** file.

To prepare the ONCONFIG file using the Instance Manager

1. Use the Instance Manager to create a new database server instance.

   For details, see "Using the Instance Manager to Create a New Database Server Instance (Windows)."
2. Use a text editor to edit the ONCONFIG configuration file.
3. Set your **ONCONFIG** environment variable to the name of your new **ONCONFIG** file.
4. Shut down and restart the database server for the configuration changes to take effect.
   a. From the **Service** control application window, select the Dynamic Server service and click the **Stop** button.
   b. Click the **Start** button.

To use dbservername.cmd to change the ONCONFIG environment variable

1. Change the **ONCONFIG** environment variable for your session in the **%INFORMIXDIR%\dbservername.cmd** file.
2. Execute ***dbservername*.cmd** for the changes to take effect.

Using the Instance Manager to Create a New Database Server Instance (Windows)

You can use the Instance Manager to create a new instance of the database server. The Instance Manager automatically creates an **ONCONFIG** file for you. For more information about the Instance Manager and scheduling priority, see *IBM Informix Dynamic Server Installation Guide for Windows*.

To use the Instance Manager

1. Select **Server Instance Manager** from the **Dynamic Server** menu.
2. Click **Create New** to create a new instance of the database server and follow the instructions in the wizard.
3. Click **Delete Server** to delete a database server instance.

Using the Instance Manager to Rename a Database Server Instance (Windows)

You can also use the Instance Manager rename option, to change the name for a database server instance.

To rename a database server instance

1. Shut down the database server instance that you want to rename.
2. Select **Server Instance Manager** from the **Dynamic Server** menu.

   The Instance Manager displays a list of available instances.
3. Select the instance to rename and click **Rename Server**.
4. In the dialog box that appears, enter your password and specify the new database server name.

After you rename the server instance, you can start the instance.

If an administrator changed the name of ONCONFIG file or the MSGLOG file to a file with a custom name before the server instance was renamed, you must manually edit the file to change the name.

Configuring Java Support

IBM Informix Dynamic Server with J/Foundation allows you to develop and run Java UDRs. Configure the database server without Java and then modify it to add Java support. Configuring the database server to support Java requires several additional steps:

To configure the database server to support Java user-defined routines

1. Create an sbspace to hold the Java JAR files.
2. Create the JVP properties file.
3. Add (or modify) the Java configuration parameters in the **ONCONFIG** file.
4. Set environment variables.

For the setup instructions, see *J/Foundation Developer's Guide*.

Start and Administer the Database Server

After you install and configure the database server, you need to perform one or more of the following tasks:

- Prepare to connect to applications.
- Start the database server and initialize disk space.
- Create storage spaces.
- Set up your backup and restore system.
- Perform administrative tasks.

Starting the Database Server and Initializing Disk Space

UNIX Only:

To bring the database server to online mode on UNIX, enter **oninit**.

If starting a new database server on UNIX, use the **oninit** command with the **-i** flag to initialize the disk space and bring the database server online.

Windows Only:

On Windows, the database server runs as a service. Use the **Service** control application to bring the database server to online mode. To initialize the database server, click the Dynamic Server service and enter **-iy** in the **Startup Parameters** box.

Another way to initialize the database server on Windows is to use the following command where *dbservername* is the database server name:

```
starts dbservername -iy
```

Information messages and error messages generated by the **oninit** command are written to the online log or to the console log. In certain situations, however, such as when using the **-v** (verbose) option or when you want to see output from an unhandled exception in a process launched within a virtual processor, set the **ONINIT_STDOUT** environment variable. When the value of **ONINIT_STDOUT** is set to the name of a file, output from the **oninit** command is written to the file. For

example, to direct output from the **oninit** command to `C:\temp\oninit_out.txt`, set the **ONINIT_STDOUT** environment variable to `C:\temp\oninit_out.txt`.

Set the **ONINIT_STDOUT** environment variable as a system variable in **Control Panel > System > Advanced > Environment Variables**. If the IDS service is configured to log on as user **informix**, start the service using the **starts** command after setting the environment variable. Note, however, that because environment variables are read from the system when the service is started, if the service is set to log on as the local system user, you must restart your computer for the environment variable to take effect. Because the local system user is effectively logged on at all times, environment variables are refreshed only when the operating system is restarted.

With the environment variable set, if the server is started with the verbose flag, output is written to `C:\temp\oninit_out.txt`:

```
starts %INFORMIXSERVER% -v
```

See the *IBM Informix Guide to SQL: Reference* for more information about **ONINIT_STDOUT**.

Automatic system startup is not available via an SQL administrative command. You must start the database server according to information in this section. If you use the SQL administration API and the Scheduler, when you use **oninit** to start the database server, the server automatically executes any stored procedures scheduled to run when the server starts. For more information, see Part 6, "Automatic Monitoring and Corrective Actions."

Initialize Only When Starting a New Database Server

Warning: When you initialize disk space, all of the existing data in the database server is destroyed. Initialize disk space only when you are starting a new database server.

For a description of the initialization types and associated commands, see Chapter 4, "Initializing the Database Server," on page 4-1.

Preparing for Automatic Startup

Prepare the operating-system registry or scripts to automatically start and stop the database server. See "Preparing for Automatic Startup on Windows" or "Preparing the UNIX Startup and Shutdown Scripts" on page 1-14.

Preparing for Automatic Startup on Windows

Change the Informix password in the **Service** control application when you change the Informix password on Windows.

To start the database server automatically when Windows starts

1. From the **Service** control application window, select the Dynamic Server service and click the **Startup** button.
2. Select **Automatic** in the **Status Type** dialog box.
3. In the **Log On As** dialog box, select **This Account** and verify that `informix` is in the text box.

Preparing the UNIX Startup and Shutdown Scripts

You can modify the UNIX startup script to initialize the database server automatically when your computer enters multiuser mode. You can also modify your UNIX shutdown script to shut down the database server in a controlled manner whenever UNIX shuts down.

To prepare the UNIX startup script, add UNIX and database server utility commands to the UNIX startup script so that the script performs the following steps.

ISA provides a sample UNIX script for startup and shutdown that you can customize in **$INFORMIXDIR/etc/ids-example.rc**.

To prepare the UNIX startup script

1. Set the **INFORMIXDIR** environment variable to the full path name of the directory in which the database server is installed.
2. Set the PATH environment variable to include the **$INFORMIXDIR/bin** directory.
3. Set the **ONCONFIG** environment variable to the desired configuration file.
4. Set the **INFORMIXSERVER** environment variable so that the **sysmaster** database can be updated (or created, if needed).
5. Execute **oninit**, which starts the database server and leaves it in online mode.

   The **oninit** utility has a **-w** option that forces the server to wait until it successfully initializes before it returns to the shell prompt with a return code of `0`. For information about the **oninit** utility, see the *IBM Informix Dynamic Server Administrator's Reference*.

   If you plan to initialize multiple versions of the database server (multiple residency), you must reset **ONCONFIG** and **INFORMIXSERVER** and re-execute **oninit** for each instance of the database server.
6. If you are using IBM Informix Storage Manager (ISM) for managing database server backups, you must start the ISM server on each node.

   For information about how to start the ISM server, see your *IBM Informix Installation Guide*.

If different versions of the database server are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each different version.

To shut down the database server in a controlled manner whenever UNIX shuts down, add UNIX and database server utility commands to the UNIX shutdown script so that the script performs the following steps.

To prepare the UNIX shutdown scrip

1. Set the **INFORMIXDIR** environment variable to the full path name of the directory in which the database server is installed.
2. Set the **PATH** environment variable to include the **$INFORMIXDIR/bin** directory.
3. Set the **ONCONFIG** environment variable to the desired configuration file.
4. Execute **onmode -ky**, which initiates an immediate shutdown and takes the database server offline.

If you are running multiple versions of the database server (multiple residency), you must reset **ONCONFIG** and re-execute **onmode -ky** for each instance.

If different versions of the database server are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each version.

In the UNIX shutdown script, the database server shutdown commands should execute after all client applications have completed their transactions and exited.

Preparing to Connect to Applications

When the database server is online, you can connect client applications and begin to create databases. Before you can access information in a database, the client application must connect to the database server environment. To connect to and disconnect from a database server, you can issue SQL statements from the following client programs:

- DB–Access
- SQL Editor
- IBM Informix ESQL/C
- IBM Informix ODBC Driver
- IBM Informix JDBC Driver

For information about creating databases, see *IBM Informix Database Design and Implementation Guide* and *IBM Informix Guide to SQL: Tutorial*. For information about how to use client applications, see the *IBM Informix DB–Access User's Guide*, *IBM Informix ESQL/C Programmer's Manual*, *IBM Informix ODBC Driver Programmer's Manual*, or *IBM Informix JDBC Driver Programmer's Guide*.

Creating Storage Spaces and Chunks

You are responsible for planning and implementing the storage configuration. The way you distribute the data on disks affects the performance of the database server. A *chunk* is the same as a logical volume, logical unit, or regular file that has been assigned to the database server. The maximum size of an individual chunk is 4 terabytes. The number of allowable chunks is 32,766. A logical *storage space* is composed of one or more chunks.

Tip: To take advantage of the large limit of 4 terabytes per chunk, assign a single chunk per disk drive. This way of distributing data increases performance.

After the database server is initialized, you can create storage spaces such as dbspaces, blobspaces, or sbspaces. Use the **onspaces** utility or ISA to create storage spaces and chunks.

You must create an sbspace if you are using the following functions:

- J/Foundation (to hold Java JAR files)
- Enterprise Replication (to hold spooled row data)
- Smart large objects (BLOB and CLOB data types)
- Multirepresentational data types (such as DataBlades or HTML data types)

For a description of storage spaces and other physical units such as tblspaces and extents, see Chapter 9, "Data Storage," on page 9-1. For a discussion of the

allocation and management of storage spaces, see “Disk-Space Parameters” on page 2-1 and Chapter 10, “Managing Disk Space,” on page 10-1.

For more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide* and *J/Foundation Developer's Guide*.

Supporting Large Chunks

If you just converted from a version of Dynamic Server that is prior to Version 9.4, chunks that are greater than 2 gigabytes are not yet enabled. To support large chunks and large offsets to the maximum size of 4 terabytes and more than 2047 chunks, run **onmode -BC 1.**

You can test your data in the **onmode -BC 1** mode. When you are satisfied that your data has converted correctly, then you can run **onmode -BC 2** and thereby put the server in large -chunk-only mode.

After running **onmode -BC 2**, reversion is no longer supported. After support for large chunks is enabled, it cannot be disabled.

Setting Up Your Backup System and Storage

To back up and restore your data, choose either the ON–Bar or **ontape** utility. For more information on setting up and using ON–Bar or **ontape**, see “Backup and Restore Parameters” on page 2-4 and the *IBM Informix Backup and Restore Guide*.

Setting Up ontape

If you use **ontape** as your backup tool, you must set up storage devices (tape drives) before you can back up and restore data. The **ontape** utility does not require a storage manager.

Setting up Your Storage Manager and ON-Bar

If you use ON–Bar as your backup tool, you must set up a storage manager and storage devices before you can back up and restore data.

ON–Bar is packaged with IBM Informix Storage Manager (ISM). The *storage manager* is an application that manages the storage devices and media that contain backups. The storage manager handles all media labeling, mount requests, and storage volumes. ISM can back up data to as many as four storage devices at a time. ISM stores data on simple tape drives, optical disk devices, and file systems. However, you can purchase a storage manager from another vendor if you want to use more sophisticated storage devices, backups to more than four storage devices at a time, or backups over a network.

When you plan your storage-space and logical-log backup schedule, make sure that the storage devices and backup operators are available to perform backups. For information about configuring and managing storage devices and media, see the *IBM Informix Storage Manager Administrator's Guide* or to your vendor-acquired storage manager documentation.

Configure Session Properties

You can change the properties of a database server session at connection or access time without changing the application that the session runs. This is useful if you cannot modify the source code of an application to set environment options or environment variables or to include session-related SQL statements, for example, because the SQL statements contain vendor-acquired code.

To change the properties of a session, design custom **sysdbopen( )** and **sysdbclose( )** procedures for various databases to support the applications of specific users or the PUBLIC group. The **sysdbopen( )** and **sysdbclose( )** procedures can contain a sequence of SET, SET ENVIRONMENT, SQL, or SPL statements that the database server executes for the user or the PUBLIC group when the database opens or closes.

For example, for *user1*, you can define procedures that contain SET PDQPRIORITY, SET ISOLATION LEVEL, SET LOCK MODE, SET ROLE, or SET EXPLAIN ON statements that execute whenever *user1* opens the database with a DATABASE or CONNECT TO statement.

Any settings of the session environment variables PDQPRIORITY and OPTCOMPIND that are specified by SET ENVIRONMENT statements within **sysdbopen( )** procedures persist for the duration of the session. SET PDQPRIORITY and SET ENVIRONMENT OPTCOMPIND statements, which are not persistent for regular procedures, are persistent when **sysdbopen( )** procedures contain them.

The *user*.**sysdbclose( )** procedure runs when the user who is the owner of the procedure disconnects from the database (or else when **PUBLIC.sysdbclose( )** runs, if it exists and no **sysdbclose( )** procedure is owned by the current user) .

In custom **sysdbopen( )** and **sysdbclose( )** procedures, Dynamic Server does not ignore the name of the owner of a UDR when a routine is invoked in a database that is not ANSI-compliant.

For more information, see "Configuring Session Properties" and the *IBM Informix Guide to SQL: Syntax*.

Configuring Session Properties

You can set up **sysdbopen( )** procedures that change the properties of a session at connection or access time without changing the application that the session runs. This is useful if you cannot modify the source code of an application to set environment options or environment variables or to include session-related SQL statements, for example, because the SQL statements contain vendor-acquired code.

Prerequisite: Only a DBA or user **informix** can create or alter **sysdbopen( )** or **sysdbclose( )** in the ALTER PROCEDURE, ALTER ROUTINE, CREATE PROCEDURE, CREATE PROCEDURE FROM, CREATE ROUTINE FROM, DROP PROCEDURE, or DROP ROUTINE statements of SQL.

To set up a sysdbopen() and sysdbclose() procedure to configure session properties

1. Set the IFX_NODBPROC environment variable to any value, including 0, to cause the database server to bypass and prevent the execution of the **sysdbopen( )** or **sysdbclose( )** procedure.
2. Write the CREATE PROCEDURE or CREATE PROCEDURE FROM statement to define the procedure for a particular user or the PUBLIC group.
3. Test the procedure, for example, by using **sysdbclose( )** in an EXECUTE PROCEDURE statement.
4. Unset the IFX_NODBPROC environment variable to enable the database server to run the **sysdbopen( )** or **sysdbclose( )** procedure.

Examples

The following procedure sets the role and the PDQ priority for a specific user:

```
create procedure oltp_user.sysdbopen()
    set role to oltp;
    set pdqpriority 5;
end procedure;
```

The following procedure sets the role and the PDQ priority for the PUBLIC group:

```
create procedure public.sysdbopen()
    set role to others;
    set pdqpriority 1;
end procedure
```

For more information, see information on **sysdbopen()** and **sysdbclose()** in the *IBM Informix Guide to SQL: Syntax*

Perform Routine Administrative Tasks

Depending on the needs of your organization, you might be responsible for performing the periodic tasks described in the following paragraphs. Not all of these tasks are appropriate for every installation. For example, if your database server is available 24 hours a day, 7 days a week, you might not bring the database server to offline mode, so database server operating mode changes would not be a routine task.

Changing Database Server Modes

The database server administrator is responsible for starting up and shutting down the database server by changing the mode. "Database Server Operating Modes" on page 4-7 explains how to change database server modes.

Backing Up Data and Logical-Log Files

To ensure that you can recover your databases in the event of a failure, make frequent backups of your storage spaces and logical logs. You also can verify ON–Bar backups with the **archecker** utility.

How often you back up the storage spaces depends on how frequently the data is updated and how critical it is. A backup schedule might include a complete (level-0) backup once a week, incremental (level-1) backups daily, and level-2 backups hourly. You also need to perform a level-0 backup after performing administrative tasks such as adding a dbspace, deleting a logical-log file, or enabling mirroring.

Back up each logical-log file as soon as it fills. You can back up these files manually or automatically. For information on using ON–Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

Monitoring Activity

The IBM Informix database server design lets you monitor every aspect of the database server. "Monitor Database Server Activity" on page 1-22 provides descriptions of the available information, instructions for obtaining information, and suggestions for its use.

Checking for Consistency

You should perform occasional checks for data consistency. For a description of these tasks, see Chapter 24, "Consistency Checking," on page 24-1.

Perform Additional Administrative Tasks

This section covers various administrative tasks that you would perform on a production database server.

Using Mirroring

When you use disk mirroring, the database server writes data to two locations. Mirroring eliminates data loss due to storage device failures. If mirrored data becomes unavailable for any reason, the mirror of the data is available immediately and transparently to users. For information on mirroring, see Chapter 17, "Mirroring," on page 17-1. For instructions on tasks related to mirroring, see Chapter 18, "Using Mirroring," on page 18-1.

Important: You should mirror critical *dbspaces* that contain logical-log files, the physical log, and the root dbspace.

Managing Database-Logging Status

You can specify whether each database uses transaction logging by default, whether the default logging mode for databases is buffered or unbuffered, and whether the logging mode is ANSI compliant.

You can create the following table types in a logging database:

- STANDARD
- TEMP
- RAW

For more information, see "Temporary Tables" on page 9-25 and Chapter 11, "Logging," on page 11-1. For information on how to change logging options, see Chapter 12, "Managing the Database-Logging Mode," on page 12-1.

Managing the Logical Log

The database server contains several files called *logical logs* that record data transactions and administrative information such as checkpoint records and additions and deletions of chunks.

Typical logical-log administration tasks include backing up logical-log files, adding, freeing, and resizing logical-log files, and specifying high-watermarks.

The database server dynamically allocates logical-log files while online to prevent long transactions from hanging the database server.

For more information, see "Logging Parameters" on page 2-2 and Chapter 13, "Logical Log," on page 13-1. For instructions on creating and modifying the logical-log configuration, see Chapter 14, "Managing Logical-Log Files," on page 14-1. For information on backing up the logical log, see the *IBM Informix Backup and Restore Guide*.

Managing the Physical Log

You can change the size and location of the physical log. For more information about the physical log, see "Physical Log Parameters" on page 2-3. Chapter 15, "Physical Logging, Checkpoints, and Fast Recovery," on page 15-1, and Chapter 16, "Managing the Physical Log," on page 16-1.

When the database server starts up, it checks whether the physical log is empty, because that implies that it shut down in a controlled fashion. If the physical log is *not* empty, the database server automatically performs an operation called *fast recovery*. Fast recovery automatically restores databases to a state of physical and logical consistency after a system failure that might have left one or more transactions uncommitted.

Managing Shared Memory

Managing shared memory includes the following tasks:

- Changing the size or number of buffers (by changing the size of the logical-log or physical-log buffer, or changing the number of buffers in the shared-memory buffer pool)
- Changing shared-memory parameter values, if necessary
- Changing forced residency (on or off, temporarily or for this session)
- Tuning checkpoint intervals
- Adding segments to the virtual portion of shared memory
- Using the SQL statement cache to reduce memory usage and preparation time for queries

For information on how the database server uses shared memory, see Chapter 7, "Shared Memory," on page 7-1. For additional information, see "Shared-Memory Parameters" on page 2-4 and Chapter 8, "Managing Shared Memory," on page 8-1.

Managing Virtual Processors

The configuration and management of virtual processors (VPs) has a direct impact on the performance of a database server. The optimal number and mix of VPs for your database server depends on your hardware and on the types of applications that your database server supports.

For an explanation of virtual processors, see Chapter 5, "Virtual Processors and Threads," on page 5-1. For additional information, see "Virtual Processor Parameters" on page 2-7 and Chapter 6, "Managing Virtual Processors," on page 6-1.

Managing Parallel Database Query

You can control the resources that the database uses to perform decision-support queries in parallel. You need to balance the requirements of decision-support queries against those of online transaction processing (OLTP) queries. The resources that you need to consider include shared memory, threads, temporary table space, and scan bandwidth. For information on parallel database query (PDQ) and how fragmentation affects the performance of PDQ, see your *IBM Informix Dynamic Server Performance Guide*. For information on the configuration parameters, see "Decision-Support Parameters" on page 2-6.

Using Data Replication

Data replication refers to the process of representing database objects at more than one distinct site. Data replication configurations consist of a primary server and one or more secondary servers, such as an HDR secondary server, one or more SD secondary servers, and one or more RS secondary servers. In addition, Dynamic Server supports *IBM Informix Enterprise Replication* (ER). You can combine data replication and Enterprise Replication on the same database server. For more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Data Replication Environments

A primary server coupled with an HDR secondary server supports synchronous replication of an entire database to the secondary database server, providing a hot standby in case of a catastrophic computer failure. For more information, see "High-Availability Data-Replication Parameters" on page 2-8. Chapter 19, "Data Replication Overview (Enterprise/Workgroup Editions)," on page 19-1, and Chapter 20, "Using High-Availability Data Replication (Enterprise/Workgroup Editions)," on page 20-1.

A primary server coupled with an SD secondary server provides read-only access to data on a disk array shared with the primary server. A primary server coupled with an RS secondary server supports asynchronous replication of the database on the primary server. A primary server coupled with any combination of HDR secondary, RS secondary, and SD secondary servers can exist in a data replication environment. See Chapter 21, "Using Remote Standalone Secondary Servers (Enterprise Edition)," on page 21-1 and Chapter 22, "Using Shared Disk Secondary Servers (Enterprise Edition)," on page 22-1.

Enterprise Replication

Enterprise Replication supports asynchronous data replication over geographically distributed database servers and allows you to replicate both entire databases and subsets of databases and tables. Enterprise Replication offers limited support of user-defined data types. For detailed information on Enterprise Replication, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Using Auditing

If you intend to use database server auditing, you need to specify where audit records are stored, how to handle error conditions, and so on. You also might want to change how users are audited if you suspect that they are abusing their access privileges. For information on tasks related to auditing, see "Auditing Parameters (UNIX)" on page 2-10 and the *IBM Informix Security Guide*.

Using Distributed Queries

The database server allows you to query and update more than one database across multiple database servers of the same database server instance (cross-database distributed queries) and across multiple server instances (cross-server distributed queries). This type of query is called a *distributed query*. This term is not restricted to SELECT statements, and often refers more generally to any DML operation or execution of a routine that returns or references objects outside the local database. In cross-server distributed queries, the participating database servers can reside on a single host computer, on the same network, or on a gateway.

For more information on distributed queries, see the *IBM Informix Database Design and Implementation Guide* and *IBM Informix Guide to SQL: Tutorial*.

For information on using the Secure Sockets Layer (SSL) protocol for connections between servers involved in distributed queries, see the ″Secure Sockets Layer Communication Protocol″ section of the *IBM Informix Security Guide*.

Using Global Transactions

A *global transaction* is a transaction that involves more than one database server. IBM Informix database servers support two types of global transactions:

- TP/XA with a transaction manager
- Two-phase commit

Informix uses a *two-phase commit protocol* to ensure that distributed queries are uniformly committed or rolled back across multiple database servers. For more information, see Chapter 25, "Multiphase Commit Protocols," on page 25-1.

Using a Transaction Manager

Transaction managers manage terminals and data recovery. The TP/XA library enables communication between a vendor-acquired transaction manager and IBM Informix databases in an X/Open environment. For more information, see the *IBM Informix TP/XA Programmer's Manual* and "Transaction Managers" on page 25-1.

Monitor Database Server Activity

You can gather information about database server activity from the following sources.

Source of Information	UNIX	Windows
Event alarms	X	X
IBM Informix Server Administrator (ISA)	X	X

Source of Information	UNIX	Windows
Message log	X	X
ON–Monitor	X	
oncheck utility	X	X
onperf utility	X	
onstat utility	X	X
SMI tables	X	X
System console	X	X
Windows Event Viewer		X
Windows Performance Logs and Alerts		X

The following sections explain each of these sources.

Event Alarms

To report situations that require your immediate attention, the database server uses the event-alarm feature. To use the event-alarm feature, set the ALARMPROGRAM configuration parameter to the full path name of an executable file that performs the necessary administrative actions.

For more information, see the appendix on event alarms and the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

IBM Informix Server Administrator (ISA)

ISA is a browser-based tool that provides system administration for the entire range of IBM Informix database servers. ISA provides access to almost all IBM Informix database server command-line functions.

For more information on ISA, the ISA online help and screen instructions.

Message Log

The database server *message log* is an operating-system file. The messages contained in the database server message log do not usually require immediate action. To report situations that require your immediate attention, the database server uses the event-alarm feature. See "Event Alarms." You can view the message log in ISA.

Specifying the Destination for Message-Log Messages

To specify the message-log path name, set the MSGPATH configuration parameter. The changes to MSGPATH take effect after you shut down and restart the database server. For more information about MSGPATH, see the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

Monitoring the Message Log

You should monitor the message log once or twice a day to ensure that processing is proceeding normally and that events are being logged as expected. Use the

onstat -m command to obtain the name of the message log and the 20 most recent entries. Use a text editor to read the complete message log. Use an operating-system command (such as the UNIX command **tail -f)** to see the messages as they occur.

Monitor the message-log size, because the database server appends new entries to this file. Edit the log as needed, or back it up to tape and delete it.

If the database server experiences a failure, the message log serves as an audit trail for retracing the events that develop later into an unanticipated problem. Often the database server provides the exact nature of the problem and the suggested corrective action in the message log.

You can read the database server message log for a minute-by-minute account of database server processing in order to catch events before a problem develops. However, you do not need to perform this kind of monitoring.

For more information, see the chapter on the messages in the *IBM Informix Dynamic Server Administrator's Reference*. Also, see *IBM Informix Error Messages*.

ON-Monitor (UNIX)

ON–Monitor provides a simple way to monitor many aspects of the database server. Most of the monitoring functions are available under the **Status** menu. For more information, see the section about ON–Monitor in the *IBM Informix Dynamic Server Administrator's Reference*.

oncheck Utility

The **oncheck** utility displays information about the database disk configuration and usage, such as the number of pages used for a table, the contents of the reserved pages, and the number of extents in a table. For more information about **oncheck**, see the *IBM Informix Dynamic Server Administrator's Reference*.

onperf Tool (UNIX)

The database server includes a graphical monitoring tool called **onperf**. This tool can monitor most of the metrics that **onstat** provides. It provides the following advantages over **onstat**:

- It displays the values of the metrics graphically in real time.
- It lets you choose which metrics to monitor.
- It saves recent-history metrics data to a buffer in memory. This data is available if you want to analyze a recent trend.
- It can save performance data to a file.

For more information about the **onperf** tool, see your *IBM Informix Dynamic Server Performance Guide*.

onstat Utility

The **onstat** utility provides a way to monitor database server shared memory from the command line. The **onstat** utility reads data from shared memory and reports statistics that are accurate for the instant during which the command executes.

That is, **onstat** provides information that changes dynamically during processing, including changes in buffers, locks, indexes, and users.

SMI Tables

The *system-monitoring interface* (SMI) tables are special tables managed by the database server that contain dynamic information about the state of the database server. You can use SELECT statements on them to determine almost anything you might want to know about your database server. For a description of the SMI tables, see the chapter about the **sysmaster** database in the *IBM Informix Dynamic Server Administrator's Reference*.

System Console

The database server sends messages that are useful to the database server administrator by way of the *system console*. To specify the destination path name of console messages, set the CONSOLE configuration parameter. For more information about CONSOLE, see the chapter about configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

The changes to CONSOLE take effect after you shut down and restart the database server.

Windows Only:

A database server system administrator can log in to the console from any node to perform system management and monitoring tasks.

UNIX Operating-System Tools

The database server relies on the operating system of the host computer to provide access to system resources such as the CPU, memory, and various unbuffered disk I/O interfaces and files. Each operating system has its own set of utilities for reporting how system resources are used. Different operating-systems might have monitoring utilities with the same name but different options and informational displays.

The following table shows typical UNIX operating-system resource-monitor utilities. For information on how to monitor your operating-system resources, consult your system administration guide.

UNIX Utility	Description
vmstat	Displays virtual-memory statistics
iostat	Displays I/O utilization statistics
sar	Displays a variety of resource statistics
ps	Displays active process information
cron	Captures the status of system resources by a system scheduler that runs a command or program at regular intervals. You also can use other scheduling tools that are available with your operating system.

Windows Event Viewer

The Event Viewer shows informational, warning, and error messages for the operating system, other applications, and the database server.

To display database server messages on Windows

1. Choose **Administrative Tools > Event Viewer**.
2. Choose **Security**.
3. Double-click any event for a more detailed message.

Windows Performance Logs and Alerts

The Windows Performance Logs and Alerts (**perfmon.exe**) monitors resources such as processor, memory, cache, threads, and processes. The Performance Logs and Alerts also provides charts, alerts, report capabilities, and the ability to save information to log files for later analysis.

To display the Performance Logs and Alerts on Windows, choose **Administrative Tools > Performance**.

Windows Utilities

The following Informix utilities simplify administration of the database server on Windows.

Windows Utility	Description and Usage
ixpasswd.exe	Changes the logon password for all services which log on as user **informix**. You can change the password interactively or on the command line using the **-y** option. This utility saves having to manually change the password for each service whenever you change the **informix** password. If you are logged on locally and run **ixpasswd**, it changes the password for services that log on as the local **informix** user. If you are logged on *domain* and run **ixpasswd**, it changes the password for services that log on as *domain***informix**. Usage: ixpasswd [-y *new_password*]
ixsu.exe	Launches a command-line window that runs as the specified user. The user is a local user unless you specify a domain name. If you do not specify a user name, the default user is **informix**. You no longer need to log off as the current user and log on as **informix** to perform DBA tasks that need to be run as **informix**. The **ixsu** utility requires Advanced User Rights: • Act as part of the operating system • Increase quotas • Replace a process-level token To configure Advanced User Rights on Windows NT®, select **User Manager > Policies > User Rights** and check the Advanced User Rights option. If you change the Advanced User Rights for the current user, you need to log off and log back on for the new rights to take effect. Usage: ixsu [[*domain*\]*username*]

Windows Utility	Description and Usage
ntchname.exe	Changes the registry entries for Dynamic Server from the old hostname to the new hostname. Run **ntchname** after you change the hostname. This utility does not change user environment variables. After you execute **ntchname**, edit the **%INFORMIXDIR%\%INFORMIXSERVER%.cmd** file and change the **INFORMIXSQLHOSTS** entry to the new hostname. Usage: ntchname *old_name new_name*

The OpenAdmin Tool for IDS

A PHP-based Web browser administration tool, the OpenAdmin Tool for IDS, provides the ability to administer multiple database server instances from a single location. The graphical user interface eases many common administration tasks, and when possible provides recommendations for tuning your system. You can complete many of the administration tasks in the following list, using the OpenAdmin Tool.

- System health monitoring using a graphical dashboard, alerts, and online message logs
  - Part 6, "Automatic Monitoring and Corrective Actions"
  - Chapter 29, "The command_history Table," on page 29-1
  - "Message Log" on page 1-23
- Database server administration
  - "Setting Permissions, Ownership, and Group" on page 1-5
  - "Preparing the ONCONFIG Configuration File" on page 1-10
  - "Monitor Database Server Activity" on page 1-22
  - Automate UPDATE STATISTICS
  - "Managing Virtual Processors" on page 1-20
  - Monitoring high availability solutions that include HDR, shared disk secondary servers, and remote standalone secondary servers: "Data Replication" on page 19-1
- Space administration
  - Chapter 10, "Managing Disk Space," on page 10-1
  - "Managing Dbspaces" on page 10-7
  - "Creating Storage Spaces and Chunks" on page 1-15
  - "Monitoring Checkpoint Information" on page 16-4
  - "Forcing a Checkpoint" on page 16-4
  - "Managing the Logical Log" on page 1-19
  - "Managing the Physical Log" on page 1-20
  - "Monitoring Physical and Logical-Logging Activity" on page 16-2
  - "Configure Disk Space" on page 1-4
- Performance gathering and analysis such as query drill down (Chapter 32, "Query Drill-Down," on page 32-1) ; session monitoring with information about SQL, locks, threads, and memory used by each session; and other methods as described in *IBM Informix Dynamic Server Performance Guide*.
- Create, modify, and delete tasks for Chapter 30, "The Scheduler," on page 30-1
- "Displaying Databases" on page 10-33, "Tables" on page 9-21, and "Extents" on page 9-7

You can easily plug in your own extensions to OpenAdmin to create the functionality you need.

OpenAdmin is an open-source program that you can download from this Web site: https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en_US&source=swg-informixfpd. See documentation provided with the OpenAdmin Tool to learn how to begin.

Chapter 2. Configuration Parameters

In This Chapter

This chapter provides an overview of the ONCONFIG configuration parameters that the database server uses and describes different ways of monitoring configuration information. The chapter can help you decide which parameters are most crucial for your particular environment and which parameters you can defer until you are tuning the performance of your database server. For details on each parameter, see the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

Database Server Identification Parameters

Use the SERVERNUM and DBSERVERNAME parameters to provide unique identification for each instance of the database server.

Configuration Parameter	Description
DBSERVERALIASES	Specifies an alternate name or names for an instance of the database server. The maximum number of aliases is 32. For information about using **DBSERVERALIASES** to create multiple listen endpoints to which clients can connect, see "Specifying Listen and Poll Threads for the Client/Server Connection" on page 5-22.
DBSERVERNAME	Specifies the unique name of an instance of the database server. Use the DBSERVERNAME for your INFORMIXSERVER environment variable and in the **sqlhosts** information. **Warning:** Do not change the DBSERVERNAME configuration parameter without restarting the database server.
SERVERNUM	Specifies a unique integer for the database server instance. The database server uses SERVERNUM to determine the shared-memory segment addresses.

Disk-Space Parameters

The disk-space parameters control how the database server manages storage space.

Root Dbspace

The first storage space that you allocate is called the *root database space*, or *root dbspace*. It stores all the basic information that describes your database server. Use the following parameters to describe the root dbspace.

Configuration Parameter	Description
ROOTNAME	Specifies the name of the root dbspace. You can choose any descriptive name for ROOTNAME, but it is usually called **rootdbs**. For more information, see "Root Dbspace" on page 9-10.
ROOTOFFSET	Specifies an offset. For information about when to set ROOTOFFSET, see "Specifying an Offset" on page 10-2.

Configuration Parameter	Description
ROOTPATH	Specifies the path name of the storage allocated to the root dbspace. For information on how to choose and allocate the storage, see "Allocating Disk Space" on page 10-1.
ROOTSIZE	Specifies the amount of space allocated to the root dbspace. For information on how to choose an appropriate size for the root dbspace, see "Size of the Root Dbspace" on page 9-31.
TBLTBLFIRST	Specifies the first extent size of tblspace **tblspace** in kilobytes.
TBLTBLNEXT	Specifies the next extent size of tblspace **tblspace** in kilobytes

Mirror of Root Dbspace

Mirroring allows fast recovery from a disk failure while the database server remains in online mode. When mirroring is active, the same data is stored on two disks simultaneously. If one disk fails, the data is still available on the other disk. Use the following parameters to describe mirroring of the root dbspace.

Configuration Parameter	Description
MIRROR	Defines whether mirroring is enabled or disabled. For more information, see Chapter 18, "Using Mirroring," on page 18-1.
MIRRORPATH	Specifies the full path name of the chunk that mirrors the initial chunk of the root dbspace.
MIRROROFFSET	Specifies the offset into the device that serves as the mirror for the initial root dbspace chunk. For more information, see "Specifying an Offset" on page 10-2.

Other Space-Management Parameters

Use the following parameters to specify how the database server should manage particular types of disk space.

Configuration Parameter	Description
DBSPACETEMP	Specifies a list of dbspaces that the database server can use for the storage of temporary tables. For more information, see "Creating a Temporary Dbspace" on page 10-15.
FILLFACTOR	Specifies how much to fill index pages when indexes are created. For more information, see your *IBM Informix Dynamic Server Performance Guide*.
ONDBSPACEDOWN	Defines how the database server treats a disabled dbspace that is not a critical dbspace.
SBSPACENAME	Specifies the name of the default sbspace. For more information, see "Control of Where Data Is Stored" on page 9-8.
SBSPACETEMP	Specifies the name of the default temporary sbspace. For more information, see "Temporary Sbspaces" on page 9-18.
SYSSBSPACENAME	Specifies the name of the sbspace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types.

Logging Parameters

Use the logging parameters to control the logical and physical logs.

Logical Log

The logical log contains a record of changes made to a database server instance. The logical-log records are used to roll back transactions, recover from system failures, and so on. The following parameters affect logical logging.

Configuration Parameter	Description
DYNAMIC_LOGS	Determines whether the database server allocates new logical-log files automatically. For more information, see Chapter 13, "Logical Log," on page 13-1.
LOGBUFF	Determines the amount of shared memory reserved for the buffers that hold the logical-log records until they are flushed to disk. For information on how to tune the logical-log buffer, see "Logical-Log Buffer" on page 7-10.
LOGFILES	Specifies the number of logical-log files used to store logical-log records until they are backed up on disk. For more information, see "Estimating the Size and Number of Log Files" on page 14-1.
LOGSIZE	Specifies the size of each logical-log file.
LTXHWM	Specifies the percentage of the available logical-log space that, when filled, triggers the database server to check for a long transaction. For more information, see "Setting High-Watermarks for Rolling Back Long Transactions" on page 14-16.
LTXEHWM	Specifies the point at which the long transaction being rolled back is given exclusive access to the logical log.
TEMPTAB_NOLOG	Disables logging on temporary tables.

Physical Log Parameters

The physical log contains images of all pages (units or storage) changed since the last checkpoint. The physical log combines with the logical log to allow fast recovery from a system failure. Use the following parameters to describe the physical log.

Configuration Parameter	Description
PHYSBUFF	Determines the amount of shared memory reserved for the buffers that serve as temporary storage space for pages about to be modified.
PHYSFILE	Specifies the size of the initial physical log when the database server is initially created. After you use the **oninit -i** command to initialize the disk space and bring the database server online, use the **onparams** utility to change the physical log location and size.

Rollback and Recovery Parameters

Information on using the following parameters, which affect rollback and recovery, is in the *IBM Informix Dynamic Server Performance Guide*. Also see Chapter 15, "Physical Logging, Checkpoints, and Fast Recovery," on page 15-1.

Configuration Parameter	Description
RTO_SERVER_RESTART	Enables you to use recovery time objective (RTO) standards to set the amount of time, in seconds, that Dynamic Server has to recover from a problem after you restart Dynamic Server and bring the server into online or quiescent mode.

Backup and Restore Parameters

Use ON–Bar or **ontape** to create storage-space and logical-log backups of database server data. To verify storage-space backups, use ON–Bar. For more information on ON–Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

If you use the **ontape** utility, use the following parameters to describe your tape devices. To use a tape to its full physical capacity, set TAPESIZE and LTAPESIZE to `0` to read/write to the end of the medium.

Configuration Parameter	Description
TAPEDEV	Specifies tape devices.
TAPEBLK	Specifies the block size of the tape device.
TAPESIZE	Specifies the maximum amount of data that should be written to each tape.
LTAPEDEV	Specifies tape devices.
LTAPEBLK	Specifies the block size of the tape device.
LTAPESIZE	Specifies the maximum amount of data that should be written to each tape.

Message-Log Parameters

The message files provide information about how the database server is functioning.

Configuration Parameter	Description
CONSOLE (UNIX)	Specifies the path name for console messages. For additional information, see "System Console" on page 1-25.
MSGPATH	Specifies the path name of the database server message-log file. For more information, see "Message Log" on page 1-23.

Shared-Memory Parameters

The shared-memory parameters affect database server performance.

Shared-Memory Size Allocation

Use the following parameters to control how and where the database server allocates shared memory.

Configuration Parameter	Description
EXTSHMADD	Specifies the size of an added extension segment.
SHMADD	Specifies the increment of memory that is added when the database server requests more memory.
SHMBASE	Specifies the shared-memory base address and is computer dependent. The value depends on the platform and whether the processor is 32-bit or 64-bit. For information on which SHMBASE value to use, see the machine notes.
SHMTOTAL	Specifies the maximum amount of memory that the database server is allowed to use.
SHMVIRTSIZE	Specifies the size of the first piece of memory that the database server attaches.

For more information on these parameters, see Chapter 7, "Shared Memory," on page 7-1.

For platform-specific information on these database server shared-memory configuration parameters, see your machine notes file on UNIX or your release notes file on Windows.

Shared-Memory Space Allocation

Use the following parameters to control how space is allocated in shared memory.

Configuration Parameter	Description
BUFFERPOOL	Specifies information about the buffer pool that must be defined for each different page size that a dbspace uses. See "Creating a Dbspace with a Non-Default Page Size" on page 10-10. Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters prior to Version 10.0 is now specified using the BUFFERPOOL configuration parameter.
CKPTINTVL	Specifies the maximum time interval allowed to elapse before a checkpoint if the RTO_SERVER_RESTART configuration parameter is not set to turn on automatic checkpoint tuning. For more information, see Chapter 15, "Physical Logging, Checkpoints, and Fast Recovery," on page 15-1.
DD_HASHMAX	Specifies the maximum number of entries for each hash bucket in the data-dictionary cache. For more information about setting DD_HASHMAX, see your *IBM Informix Performance Guide*.
DD_HASHSIZE	Specifies the number of hash buckets in the data-dictionary cache. For more information about setting DD_HASHSIZE, see your *IBM Informix Performance Guide*.
DEF_TABLE_ LOCKMODE	Sets the lock mode to page or row for new tables. For more information, see the *IBM Informix Guide to SQL: Tutorial*.
LOCKS	Specifies the initial number of locks available to database server user processes during transaction processing. For more information, see Chapter 8, "Managing Shared Memory," on page 8-1 and your *IBM Informix Dynamic Server Performance Guide*.
PC_POOLSIZE	Specifies the number of UDRs (both SPL routines and external routines) that can be stored in the UDR cache. For more information about setting PC_POOLSIZE, see your *IBM Informix Dynamic Server Performance Guide*.
PC_HASHSIZE	Specifies the number of hash buckets for the UDR cache. For more information about setting PC_HASHSIZE, see your *IBM Informix Dynamic Server Performance Guide*.
RESIDENT	Specifies whether shared-memory residency is enforced. For more information, see Chapter 8, "Managing Shared Memory," on page 8-1 and your *IBM Informix Dynamic Server Performance Guide*.
STACKSIZE	Specifies the stack size for database server user threads. For a discussion of the use of stacks, see "Stacks" on page 7-17.

Shared-Memory Buffer Control

Use the following parameters to control the shared-memory buffer pool.

Configuration Parameter	Description
CLEANERS	Controls the number of threads used to flush pages to disk and return the pages to the shared-memory pool. See "Flushing Data to Disk" on page 7-25.

Configuration Parameter	Description
RA_PAGES and RA_THRESHOLD	Control the number of disk pages that the database server reads ahead during sequential scans. See "Configuring the Database Server to Read Ahead" on page 7-25.

SQL Statement Cache Usage

Use the following parameters to configure the SQL statement cache. For more information, see "Setting SQL Statement Cache Parameters" on page 8-5.

Configuration Parameter	Description
STMT_CACHE	Turns on, enables, or disables the SQL statement cache in memory. If turned on, specifies whether the SQL statement cache can hold a parsed and optimized SQL statement.
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.
STMT_CACHE_HITS	Specifies the number of hits (references) to a statement before it is fully inserted into the SQL statement cache.
STMT_CACHE_NOLIMIT	Controls whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.
STMT_CACHE_NUMPOOL	Defines the number of memory pools for the SQL statement cache.

Decision-Support Parameters

When you configure virtual shared memory on your system, you must decide what portion to reserve for decision-support queries. Decision-support queries use large amounts of the virtual portion of shared memory to perform joins and sort operations.

Use the following parameters to control how decision-support queries are processed and to control the amount of memory that the database server allocates to decision-support queries. For more information about tuning these configuration parameters, see your *IBM Informix Dynamic Server Performance Guide*.

Configuration Parameter	Description
DATASKIP	Controls whether the database server skips an unavailable table fragment.
DS_MAX_QUERIES	Specifies the maximum number of queries that can run concurrently.
DS_MAX_SCANS	Limits the number of Parallel Database Query (PDQ) scan threads that the database server can execute concurrently.
DS_TOTAL_MEMORY	Specifies the amount of memory available for PDQ queries. Set the DS_TOTAL_MEMORY configuration parameter to any value not greater than the quantity (SHMVIRTSIZE - 10 megabytes).
DS_NONPDQ_QUERY_MEM	Enables you to increase the amount of memory that is available for a query that is not a PDQ query.
MAX_PDQPRIORITY	Limits the amount of resources that a query can use.
OPTCOMPIND	Advises the optimizer on an appropriate join strategy for your applications.

Database Server Process Parameters

Configuration parameters for database server processes describe the type of processors on your computer and specify the behavior of virtual processes.

Virtual Processor Parameters

Use the following parameters to specify the type of processors in your environment and to allocate the virtual processors.

You need to set the following parameters to specific values, depending upon the number of processors on your platform:

- AUTO_AIOVPS
- MULTIPROCESSOR
- NETTYPE
- SINGLE_CPU_VP
- VPCLASS

For guidelines on setting these parameters, see "Setting Virtual-Processor Configuration Parameters" on page 6-1.

Configuration Parameter	Description
AUTO_AIOVPS	Enables or disables the ability of the database server to automatically increase the number of AIO VPS and flusher threads when the server detects that AIO VPs are not keeping up with the I/O workload.
MULTIPROCESSOR	Specifies the appropriate type of locking.
NETTYPE	Provides tuning options for each communications protocol.
SINGLE_CPU_VP	Specifies that the database server is using only one processor and allow the database server to optimize for that situation.
VPCLASS	Specifies a class of virtual processors, the number of virtual processors that the database server should start, the maximum number allowed, the processor affinity, and the priority aging. You should use the VPCLASS parameter instead of NOAGE, NUMCPUVPS, NUMAIOVPS, AFF_NPROCS, or AFF_SPROCS. For more information, see "Specifying VPCLASS" on page 6-2.

When adding new storage dbspaces, blobspaces, and sbspaces, the database server automatically adjusts the number of AIO VPs and cleaner threads to meet the new storage demands.

Time Intervals

Use the following parameters to control the time intervals that the database server uses while processing transactions.

Configuration Parameter	Description
DEADLOCK_TIMEOUT	Specifies the amount of time that the database server waits for a shared-memory resource during a distributed transaction.
HETERO_COMMIT	Specifies whether the database server uses heterogeneous commit transactions.
TXTIMEOUT	Specifies how long a participant waits to receive a *commit* instruction during a two-phase commit.

Configuration Parameter	Description
USEOSTIME	Controls the granularity of time reported by the database server.

Restore Parameters

Use the following parameters to control the number of threads that the database server allocates to offline and online logical recovery. For more information, see your *IBM Informix Dynamic Server Performance Guide*.

Configuration Parameter	Description
OFF_RECVRY_THREADS	Specifies the number of recovery threads used during fast recovery and a cold restore.
ON_RECVRY_THREADS	Specifies the number of recovery threads used during a warm restore.

High-Availability Data-Replication Parameters

Use the High-Availability Data-replication (HDR) parameters to control the behavior of a pair of HDR servers. For more information, see Chapter 19, "Data Replication Overview (Enterprise/Workgroup Editions)," on page 19-1.

Configuration Parameter	Description
DRIDXAUTO	Specifies whether the primary High-Availability Data Replication (HDR) server automatically starts index replication if the secondary HDR server detects a corrupted index.
DRAUTO	Defines how a secondary database server reacts to an HDR failure.
DRINTERVAL	Specifies the maximum time interval in seconds between flushing of the data-replication buffer.
DRLOSTFOUND	Specifies the path name to a file that contains transactions committed on the primary database server but not committed on the secondary database server when the primary database server experiences a failure.
DRTIMEOUT	Specifies how long in seconds a database server in a data-replication pair waits for a transfer acknowledgment from the other database server in the pair.

Event-Alarm Parameters

The database server can execute a program that operates either whenever certain noteworthy event alarms occur or every time any event alarm occurs. Noteworthy event alarms include failure of a database, table, index, chunk or dbspace taken offline, internal subsystem failure, start-up failure, and detection of long transaction. You can receive notification of an event alarm through E-mail or pager mail.

Use the following parameters to specify:

- Whether the event-alarm program operates for all or only certain events alarms
- What actions to take when alarm events occur

Configuration Parameter	Description
ALRM_ALL_EVENTS	Specifies whether ALARMPROGRAM runs for all events that are logged in the MSGPATH or only specified noteworthy events.

Configuration Parameter	Description
ALARMPROGRAM	Specifies the location of a file that is executed when an event alarm occurs. You can set the ALARMPROGRAM parameter to back up logs automatically as they fill. For more information, see the *IBM Informix Backup and Restore Guide*.

For more information, see your *Administrator's Reference.*

Dump Parameters (UNIX)

Use the following parameters to control the types and location of core dumps that are performed if the database server fails. For more information, see "Monitoring for Data Inconsistency" on page 24-3.

Configuration Parameter	Description
DUMPCNT	Specifies the number of assertion failures for which a single thread dumps shared memory.
DUMPCORE	Controls whether assertion failures cause a virtual processor to dump core memory.
DUMPDIR	Specifies a directory where the database server places dumps of shared memory, **gcore** files, or messages from a failed assertion.
DUMPGCORE	If your operating system supports the **gcore** utility, an assertion failure causes the database server to call **gcore**.
DUMPSHMEM	Specifies that shared memory should be dumped on an assertion failure.

Directives Parameters

You can use the following configuration parameters to turn on or off directives that the database server encounters.

Configuration Parameter	Description
DIRECTIVES	Determines whether the optimizer follows all directives. The default value is 1, indicating the optimizer follows all directives. For more information, see the *IBM Informix Dynamic Server Performance Guide*.
EXT_DIRECTIVES	Enables an external directive. For more information, see the *IBM Informix Dynamic Server Administrator's Reference* or the *IBM Informix Dynamic Server Performance Guide*.

Connection Parameters

Use the following parameters to configure connectivity use to reduce the incomplete connection timeout period and restrict the number of incomplete requests for connections, thus reducing the risk of a hostile, denial-of-service (DOS) flood attack. For more information, see *IBM Informix Security Guide*.

Configuration Parameter	Description
LISTEN_TIMEOUT	Sets the incomplete connection timeout period. The default value is 10 seconds.
MAX_INCOMPLETE_ CONNECTIONS	Restricts the number of incomplete requests for connections. The default value is 1024.

Security-Related Parameters

The following security-related parameters are discussed more fully in the *IBM Informix Security Guide*.

Configuration Parameter	Description
DBCREATE_PERMISSION	Gives permission to a specified user to create databases, thus restricting other users from creating databases.
DB_LIBRARY_PATH	Enables a database server administrator (DBSA) to specify a comma-separated list of valid directory prefix locations from which the database server can load external modules.
IFX_EXTEND_ROLE	Enables a DBSA to prevent unauthorized users from registering DataBlade modules or external user-defined routines (UDRs).
SECURITY_ LOCALCONNECTION	Enables a DBSA to set up security checking for local connections with the same host.
SSL_KEYSTORE_ LABEL	Specifies the label of the server digital certificate used in Secure Sockets Layer (SSL) protocol communications.
UNSECURE_ONSTAT	Removes the restriction that only allows DBA users to execute **onstat -g ses**, **onstat -g stm**, **onstat -g ssc**, and **onstat -g sql** commands.

The following security-related parameters are discussed more fully in "Encrypting Data Traffic Between HDR Database Servers" on page 20-15 and the *IBM Informix Dynamic Server Administrator's Reference*.

Configuration Parameter	Description
ENCRYPT_HDR	Enables HDR encryption.
ENCRYPT_CIPHERS, ENCRYPT_MAC, ENCRYPT_MACFILE, and ENCRYPT_SWITCH	Specifies information for HDR encryption.

Specialized Parameters

Some parameters appear in the configuration file only when you use specialized features of the database server.

Auditing Parameters (UNIX)

Use the auditing parameters only when you use the database server auditing features. For more information on these parameters, see the *IBM Informix Security Guide*.

Configuration Parameter	Description
ADTERR	Specifies how the database server behaves when it encounters an error while it writes an audit record.
ADTMODE	Controls whether the database server or the operating system manages auditing of user actions.
ADTPATH	Specifies the directory in which the database server can save audit files.

Configuration Parameter	Description
ADTSIZE	Specifies the maximum size of an audit file.

Optical Media Parameters

Use the following configuration parameters when you use the Optical Subsystem. For more information about these parameters, see the *IBM Informix Optical Subsystem Guide*.

Configuration Parameter	Description
OPCACHEMAX	Specifies the size of the memory cache.
STAGEBLOB	Specifies the name of the blobspace for storing simple large objects that are destined for optical disk.

IBM Informix Storage Manager allows you to back up information to optical media, but it does not allow the database server to access directly the data that is stored on the disks.

UNIX Parameters

Some UNIX platforms have additional configuration parameters. For a description of these specialized parameters and instructions on using them, see your machine notes.

Monitoring Configuration Information

One of the tasks of the database server administrator is to keep records of the configuration. Table 2-1 describes methods of obtaining configuration information.

Table 2-1. Monitoring Configuration Information

Command	Description
onstat -c	Displays a copy of the ONCONFIG file. For more information, see "Configure the Database Server Configuration" on page 1-10. Changes to the ONCONFIG file take effect when you shut down and restart the database server, also called *reinitializing shared memory*. If you change a configuration parameter but do not shut down and restart the database server, the effective configuration differs from what the **onstat -c** option displays. The values of the configuration parameters are stored in the file indicated by the **ONCONFIG** environment variable or, if you have not set the **ONCONFIG** environment variable, in **$INFORMIXDIR/etc/onconfig** on UNIX or **%INFORMIXDIR%\etc\onconfig.std** on Windows.
oncheck -pr	Lists the reserved page. The database server also stores current configuration information in the PAGE_CONFIG reserved page. If you change the configuration parameters from the command line and run **oncheck -pr** without shutting down and restarting the database server, the configuration values that **oncheck** displays do not match the current values in the reserved pages. The **oncheck** utility returns a warning message.
ON–Monitor (UNIX)	Select **Status > Configuration** to create a copy of the current configuration and store it in the directory and file that you specify. If you specify only a filename, the database server stores the file in the current working directory. Changes to the configuration parameters take effect when you shut down and restart the database server.
ISA	Displays or updates the configuration parameters.

Figure 2-1 shows sample output from the **oncheck -pr** command.

```
...

Validating Informix database server reserved pages - PAGE_CONFIG
    ROOTNAME                        rootdbs
    ROOTPATH                        /home/dyn_srv/root_chunk
    ROOTOFFSET                      4
    ROOTSIZE                        8000
    MIRROR                          0
    MIRRORPATH
    MIRROROFFSET                    0
    PHYSFILE                        1000
    LOGFILES                        5
    LOGSIZE                         500
    MSGPATH                         /home/dyn_srv/online.log
    CONSOLE                             /dev/ttyp5
...                                 ...
```

Figure 2-1. PAGE_CONFIG Reserved Page

Chapter 3. Client/Server Communications

In This Chapter

This chapter explains the concepts and terms that you need to understand in order to configure client/server communications. The chapter consists of the following parts:

- Description of client/server architecture
- Database server connection types
- Communication services
- Connectivity files
- ONCONFIG connectivity parameters
- Connectivity environment variables
- Examples of client/server configurations

Client/Server Architecture

IBM Informix products conform to a software design model called *client/server*. The client/server model allows you to place an application or *client* on one computer and the database *server* on another computer, but they can also reside on the same computer. Client applications issue requests for services and data from the database server. The database server responds by providing the services and data that the client requested.

You use a *network protocol* together with a *network programming interface* to *connect* and transfer data between the client and the database server. The following sections define these terms in detail.

Network Protocol

A network protocol is a set of rules that govern how data is transferred between applications and, in this context, between a client and a database server. These rules specify, among other things, what format data takes when it is sent across the network. An example of a network protocol is TCP/IP.

The rules of a protocol are implemented in a *network-protocol driver.* A network-protocol driver contains the code that formats the data when it is sent from client to database server and from database server to client.

Clients and database servers gain access to a network driver by way of a *network programming interface.* A network programming interface contains system calls or library routines that provide access to network-communications facilities. An example of a network programming interface for UNIX is TLI (Transport Layer Interface). An example of a network programming interface for Windows is WINSOCK (sockets programming interface).

The power of a network protocol lies in its ability to enable client/server communication even though the client and database server reside on different computers with different architectures and operating systems.

You can configure the database server to support more than one protocol, but consider this option only if some clients use TCP/IP and some use IPX/SPX.

To determine the supported protocols for your operating system, see "Database Server Connection" on page 3-3.

To specify which protocol the database server uses, set the nettype field in the **sqlhosts** file on UNIX. On Windows, set the PROTOCOL field in the SQLHOSTS registry key. For more information, see "The sqlhosts File and the SQLHOSTS Registry Key" on page 3-13.

Network Programming Interface

A network programming interface is an application programming interface (API) that contains a set of communications routines or system calls. An application can call these routines to communicate with another application that resides on the same or on different computers. In the context of this discussion, the client and the database server are the applications that call the routines in the TLI or sockets API. Clients and database servers both use network programming interfaces to send and receive the data according to a communications protocol.

Both client and database server environments must be configured with the same protocol if client/server communication is to succeed. However, some network protocols can be accessed through more than one network programming interface. For example, TCP/IP can be accessed through either TLI or sockets, depending on which programming interface is available on the operating-system platform. Therefore, a client using TCP/IP through TLI on one computer can communicate with a database server using TCP/IP with sockets on another computer, or vice versa. For an example, see "Using a Network Connection" on page 3-42.

Windows Network Domain

Windows network technology enables you to create network *domains*. A domain is a group of connected Windows computers that share user account information and a security policy. A *domain controller* manages the user account information for all domain members.

The domain controller facilitates network administration. By managing one account list for all domain members, the domain controller relieves the network administrator of the requirement to synchronize the account lists on each of the domain computers. In other words, the network administrator who creates or changes a user account needs to update only the account list on the domain controller rather than the account lists on each of the computers in the domain.

To log in to a Windows database server, a user on another Windows computer must belong to either the same domain or a *trusted domain*. A trusted domain is one that has established a *trust relationship* with another domain. In a trust relationship, user accounts are located only in the trusted domain, but users can log on to the trusted domain.

A user who attempts to log in to a Windows computer that is a member of a domain can do so either by using a local login and profile or a domain login and profile. However, if the user is listed as a trusted user or the computer from which the user attempts to log in is listed as a trusted host, the user can be granted login access without a profile.

Important: A client application can connect to an Informix database server only if there is an account for the user ID in the Windows domain in which the database server runs. This rule also applies to trusted domains.

If you specify a user identifier but no domain name for a connection to a machine that expects both a domain name and a user name (domain\user), IDS checks only the local machine and the primary domain for the user account. If you explicitly specify a domain name, that domain is used to search for the user account. The attempted connection fails with error -951 if no matching domain\user account is found on the local machine.

Use the **CHECKALLDOMAINSFORUSER** configuration parameter to configure how IDS searches for user names in a networked Windows environment. The following table lists the locations IDS searches for user names specified either alone or with a domain name with **CHECKALLDOMAINSFORUSER** set to 0 or 1.

Table 3-1. The CHECKALLDOMAINSFORUSER Configuration Parameter (Windows)

	Domain\user specified	**User Name only specified**
CHECKALLDOMAINSFORUSER=0	IDS searches for the user name only in the specified domain.	IDS searches for the user name on the local host only.
CHECKALLDOMAINSFORUSER=1	IDS searches for the user name only in the specified domain.	IDS searches for the user name in all domains.

Omitting **CHECKALLDOMAINSFORUSER** from the ONCONFIG file is the same as setting **CHECKALLDOMAINSFORUSER** to 0. See the *IBM Informix Dynamic Server Administrator's Reference* for more information about setting **CHECKALLDOMAINSFORUSER**.

For more information on domains, consult your Windows operating system manuals.

Important: The Informix trusted client mechanism is unrelated to the trust relationship that you can establish between Windows domains. Therefore, even if a client connects from a trusted Windows domain, the user must have an account in the domain on which the database server is running. For more information on how the database server authenticates clients, see "Communication Support Services" on page 3-8 and "Network Security Files" on page 3-11.

Database Server Connection

A *connection* is a logical association between two applications; in this context, between a client application and a database server. A connection must be established between client and database server *before* data transfer can take place. In addition, the connection must be maintained for the duration of the transfer of data.

Tip: The Informix internal communications facility is called Association Services Facility (ASF). If you see an error message that refers to ASF, you have a problem with your connections.

A client application establishes a connection to a database server with either the CONNECT or DATABASE SQL statement. For example, to connect to the database server my_server, an application might contain the following form of the CONNECT statement:

```
CONNECT TO '@my_server'
```

For more information on the CONNECT and DATABASE statements, see the *IBM Informix Guide to SQL: Syntax*.

Multiplexed Connections

Some applications connect multiple times to the same database server on behalf of one user. A *multiplexed connection* uses a single *network* connection between the database server and a client to handle multiple *database* connections from the client. Client applications can establish multiple connections to a database server to access more than one database on behalf of a single user. If the connections are not multiplexed, each database connection establishes a separate network connection to the database server. Each additional network connection consumes additional computer memory and CPU time, even for connections that are not active. Multiplexed connections enable the database server to create multiple database connections without consuming the additional computer resources that are required for additional network connections.

To configure the database server to support mulitplexed connections:

1. Define an alias using the DBSERVERALIASES configuration parameter. For example, specify:

   ```
   DBSERVERALIASES ids_mux
   ```

2. Add an SQLHOSTS entry for the alias using `sqlmux` as the **nettype** entry, which is the second column in the SQLHOSTS file. For example, specify:

   ```
   ids_mux  onsqlmux  ......
   ```

   The other fields in this entry, the **hostname** and **servicename**, must be present, but they are ignored.

3. Enable multiplexing for the selected connection types by specifying `m=1` in the **sqlhosts** file or registry that the client uses for the database server connection. For example:

   ```
   menlo ontlitcp valley jfk1 m=1
   ```

4. On Windows platforms, you must also set the **IFX_SESSION_MUX** environment variable.

The following example shows connectivity formation for a situation in which the configuration file contains the following entries:

- DBSERVERNAME web_tli
- DBSERVERALIASES web_mux

```
web_tli   ontlitcp    node5    svc5   m=1
web_mux   onsqlmux    -        -
```

You do not need to make any changes to the **sqlhosts** file or registry that the database server uses. The client program does not need to make any special SQL calls to enable connections multiplexing. Connection multiplexing is enabled automatically when the ONCONFIG file and the **sqlhosts** file or SQLHOSTS registry key are configured appropriately. For more information on the **sqlhosts** file or registry, see "The sqlhosts File and the SQLHOSTS Registry Key" on page 3-13.

The following limitations apply to multiplexed connections:

- Multithreaded client connections are not supported.
- Shared-memory connections are not supported.
- Connections to subordinate database servers (for distributed queries or data replication, for example) are not multiplexed.
- The Informix ESQL/C **sqlbreak()** function is not supported.
- You can activate database server support for multiplexed connections only when the database server starts.

If any of these conditions exist when an application attempts to establish a connection, the database server establishes a standard connection. The database server does not return an SQL error.

Connections That the Database Server Supports

The database server supports the following types of connections to communicate between client applications and a database server.

Connection Type	Windows	UNIX	Local	Network
Sockets	X	X	X	X
TLI (TCP/IP)		X	X	X
TLI (IPX/SPX)		X	X	X
Shared memory		X	X	
Stream pipe		X	X	
Named pipe	X		X	

Note: When configuring connectivity, consider setting the IFX_LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTION configuration parameters. These parameters enable you to reduce the risk of a hostile denial-of-service (DOS) attack by making it more difficult to overwhelm the Listener VP that handles connections. For more information, see the *IBM Informix Security Guide*.

UNIX Only:

On many UNIX platforms, the database server supports multiple network programming interfaces. The machine notes shows the interface/protocol combinations that the database server supports for your operating system:

```
Machine Specific Notes:
=======================

1. The following interface/protocol combinations(s) are
   supported for this platform:

            Berkeley sockets using TCP/IP
```

To set up a client connection

1. Specify connectivity and connection configuration parameters in your ONCONFIG file.
2. Set up appropriate entries in the connectivity files on your platform.
3. Specify connectivity environment variables in your UNIX start-up scripts or the local and domainwide Windows registries.

4. Define a dbserver group for your database server in the **sqlhosts** file or registry.

The following sections describe database server connection types in more detail. For detailed information about implementing the connections described in the following sections, see the following topics:

- "Connectivity Files" on page 3-8
- "The sqlhosts Information" on page 3-16
- "ONCONFIG Parameters for Connectivity" on page 3-32
- "Environment Variables for Network Connections" on page 3-35

Local Connections

A *local connection* is a connection between a client and the database server on the same computer. The following sections describe these types of local connections.

Shared-Memory Connections (UNIX)

A *shared-memory connection* uses an area of shared-memory as the *channel* through which the client and database server communicate with each other. Figure 3-1 illustrates a shared-memory connection.

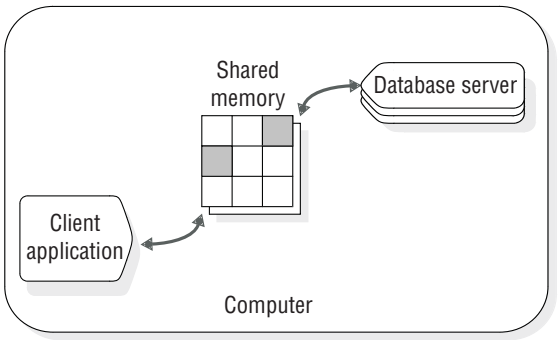


Figure 3-1. A Shared-Memory Connection

Shared memory provides fast access to a database server, but it poses some security risks. Errant or malicious applications could destroy or view message buffers of their own or of other local users. Shared-memory communication is also vulnerable to programming errors if the client application performs explicit memory addressing or overindexes data arrays. Such errors do not affect the database server if you use network communication or stream pipes. For an example of a shared-memory connection, see "Using a Shared-Memory Connection (UNIX)" on page 3-41.

A client cannot have more than one shared-memory connection to a database server.

For information about the portion of shared memory that the database server uses for client/server communications, see "Communications Portion of Shared Memory (UNIX)" on page 7-19. For additional information, you can also see "How a Client Attaches to the Communications Portion (UNIX)" on page 7-5.

Stream-Pipe Connections (UNIX)

A *stream pipe* is a UNIX interprocess communication (IPC) facility that allows processes on the same computer to communicate with each other. You can use stream-pipe connections any time that the client and the database server are on the same computer. For more information, see "Network Protocol Entry" on page 3-18 and "Shared-Memory and Stream-Pipe Communication (UNIX)" on page 3-20.

Stream-pipe connections have the following advantages:

- Unlike shared-memory connections, stream pipes do not pose the security risk of being overwritten or read by other programs that explicitly access the same portion of shared memory.
- Unlike shared-memory connections, stream-pipe connections allow distributed transactions between database servers that are on the same computer.

Stream-pipe connections have the following disadvantages:

- Stream-pipe connections might be slower than shared-memory connections on some computers.
- Stream pipes are not available on all platforms.
- When you use shared memory or stream pipes for client/server communications, the **hostname** entry is ignored.

Stream-Pipe Connections (Linux)

On Linux platforms, Dynamic Server supports Interprocess communication (IPC) using stream pipes and UNIX Domain Sockets.

For stream-pipe connections to occur, NETTYPE in the **$INFORMIXDIR/etc/$ONCONFIG** file and the **nettype** field in the **$INFORIMXDIR/etc/sqlhosts** file entry must contain `onipcstr`.

For example, when `onipcstr` is specified, local 32-bit applications and tools can connect to a 64-bit server using the IPC stream pipe protocols.

Named-Pipe Connections (Windows)

Named pipes are application programming interfaces (APIs) for bidirectional interprocess communication (IPC) on Windows. Named-pipe connections provide a high-level interface to network software by making transport-layer operations transparent. Named pipes store data in memory and retrieve it when requested, in a way that is similar to reading from and writing to a file system.

Named pipes are supported for local connections to the database server.

Local-Loopback Connections

A network connection between a client application and a database server on the same computer is called a *local-loopback* connection. The networking facilities used are the same as if the client application and the database server were on different computers. You can make a local-loopback connection provided your computer is equipped to process network transactions. Local-loopback connections are not as fast as shared-memory connections, but they do not pose the security risks of shared memory.

In a local-loopback connection, data appears to pass from the client application, out to the network, and then back in again to the database server. In fact, although the database server uses the network programming interface (TLI or sockets), the internal connection processes send the information directly between the client and the database server and do *not* put the information out on the network.

For an example of a local-loopback connection, see "Using a Local-Loopback Connection" on page 3-42.

Communication Support Services

Communication support services include connectivity-related services such as the following security services:

- Authentication is the process of verifying the identity of a user or an application. The most common form of authentication is to require the user to enter a name and password to obtain access to a computer or an application.
- Message integrity ensures that communication messages are intact and unaltered when they arrive at their destination.
- Message confidentiality protects messages, usually by encryption and decryption, from viewing by unauthorized users during transmission.

Communication support services can also include other processing such as data compression or traffic-based accounting.

The database server provides a default method of authentication, which is described in "Network Security Files" on page 3-11. The database server uses the default authentication policy when you do not specify a communications support module.

The database server provides extra security-related communication support services through plug-in software modules called Communication Support Modules (CSM). For details, see *IBM Informix Security Guide*.

Connectivity Files

The *connectivity files* contain the information that enables client/server communication. These files also enable a database server to communicate with another database server. The connectivity configuration files can be divided into three groups:

- Network-configuration files
- Network security files
- The **sqlhosts** file or SQLHOSTS registry

Network-Configuration Files

This section identifies and explains the use of network-configuration files on TCP/IP and IPX/SPX networks.

TCP/IP Connectivity Files

When you configure the database server to use the TCP/IP network protocol, you use information from the hosts and services network-configuration files to prepare the sqlhosts information.

The network administrator maintains these files. When you add a host or a software service such as a database server, you need to inform the network administrator so that person can make sure the information in these files is accurate.

The **hosts** file needs a single entry for each network-controller card that connects a computer running an Informix client/server product on the network. Each line in the file contains the following information:

- Internet address (or ethernet card IP address)
- Host name
- Host aliases (optional)

Although the length of the host name is not limited in the **hosts** file, Informix limits the host name to 256 bytes. Table 3-5 on page 3-29 includes a sample **hosts** file.

The **services** file contains an entry for each service available through TCP/IP. Each entry is a single line that contains the following information:

- Service name

  IBM Informix products use this name to determine the port number and protocol for making client/server connections. The service name can have up to 128 bytes.
- Port number and protocol

  The port number is the computer port, and the protocol for TCP/IP is `tcp`.

  The operating system imposes restrictions on the port number. User **informix** must use a port number equal to or greater than `1024`. Only **root** users are allowed to use a port number less than `1024`.
- Aliases (optional)

The service name and port number are arbitrary. However, they must be unique within the context of the file and must be identical on all computers that are running IBM Informix client/server products. The aliases field is optional. For example, a **services** file might include the following entry for a database server:

```
server2      1526/tcp
```

This entry makes server2 known as the service name for TCP port 1526. A database server can then use this port to service connection requests. Figure 3-4 on page 3-21 includes a sample **services** file.

Important: For database servers that communicate with other database servers, you must define a TCP/IP connection in DBSERVERNAME or DBSERVERALIASES even when both instances reside on the same machine.

For information about the **hosts** and **services** files, see your operating system documentation.

TCP/IP Connectivity Files on UNIX:

On UNIX, the **hosts** and **services** files are in the **/etc** directory. The files must be present on each computer that runs an IBM Informix client/server product, or on the NIS server if your network uses *Network Information Service* (NIS).

Warning: On systems that use NIS, the **/etc/hosts** and **/etc/services** files are maintained on the NIS server. The **/etc/hosts** and /etc/services files that reside on

your local computer might not be used and might not be up to date. To view the contents of the NIS files, enter the following commands on the command line:

```
ypcat hosts
ypcat services
```

TCP/IP Connectivity Files on Windows:

You use information from the **hosts** and **services** network-configuration files to prepare the SQLHOSTS registry key for the TCP/IP network protocol. These files are in the following locations:

- **%WINDIR%\system32\drivers\etc\hosts**
- **%WINDIR%\system32\drivers\etc\services**

Alternately, you can configure TCP/IP to use the Domain Name Service (DNS) for host name resolutions. For information about these files, see your operating-system documentation.

The Dynamic Host Configuration Product (DHCP) dynamically assigns IP addresses from a pool of addresses instead of using IP addresses that are explicitly assigned to each workstation. If your system uses DHCP, you must also have installed Windows Internet Name Service (WINS). DCHP is transparent to the database server.

What Happens Between a Client and Server When a TCP/IP Connection Is Opened:

When a TCP/IP connection is opened, the following information is read on the client side:

- INFORMIXSERVER
- **hosts** file information (INFORMIXSQLHOSTS, $INFORMIXDIR/etc/sqlhosts, the registry entry on Windows NT) and **services** file information
- Other environment variables
- Resource files

The following information is read on the server side:

- DBSERVERNAME
- DBSERVERALIASES
- Server environment variables and configuration parameters, including any NETTYPE configuration parameter setting that manage TCP/IP connections

  For more information about the NETTYPE configuration parameter, see "NETTYPE Configuration Parameter" on page 3-35 and the *IBM Informix Dynamic Server Administrator's Reference*.

Multiple TCP/IP Ports

To take advantage of multiple ethernet cards, take the following actions:

- Make an entry in the **services** file for each port the database server will use, as in the following example:

  ```
  soc1        21/tcp
  soc2        22/tcp
  ```

Each port of a single IP address must be unique. Separate ethernet cards can use unique or shared port numbers. You might want to use the same port number on ethernet cards connecting to the same database server. (In this scenario, the service name is the same.)

- Put one entry per ethernet card in the **hosts** file with a separate IP address, as in the following example:

```
192.147.104.19          svc8
192.147.104.20          svc81
```

- In the ONCONFIG configuration file, enter DBSERVERNAME for one of the ethernet cards and DBSERVERALIASES for the other ethernet card. The following lines show sample entries in the ONCONFIG file:

```
DBSERVERNAME chicago1
DBSERVERALIASES chicago2
```

- In the **sqlhosts** file on UNIX or the SQLHOSTS registry key on Windows, make one entry for each ethernet card. That is, make an entry for the DBSERVERNAME and another entry for the DBSERVERALIASES.

```
chicago1 onsoctcp svc8 soc1
chicago2 onsoctcp svc81 soc2
```

After this configuration is in place, the application communicates through the ethernet card assigned to the **dbservername** that the **INFORMIXSERVER** environment variable provides.

IPX/SPX Connectivity Files (UNIX)

To configure the database server to use the IPX/SPX protocol on a UNIX network, you must purchase IPX/SPX software and install it on the database server computer. Your choice of IPX/SPX software depends on the operating system that you are using. For some operating systems, the IPX/SPX software is bundled with software products based on NetWare for UNIX or Portable NetWare. In addition, for each of the UNIX vendors that distributes IPX/SPX software, you might find a different set of configuration files.

For advice on how to set configuration files for these software products, consult the manuals that accompany your IPX/SPX software.

Network Security Files

IBM Informix products follow standard security procedures that are governed by information contained in the network security files. For a client application to connect to a database server on a remote computer, the user of the client application must have a valid user ID on the remote computer.

The hosts.equiv File

The **hosts.equiv** file lists the remote hosts and users that are trusted by the computer on which the database server resides. Trusted users, and users who log in from trusted hosts, can access the computer without supplying a password. The operating system uses the **hosts.equiv** file to determine whether a user should be allowed access to the computer without specifying a password. Informix requires a **hosts.equiv** file for its default authentication policy.

If a client application supplies an invalid account name and password, the database server rejects the connection even if the **hosts.equiv** file contains an entry for the client computer. You should use the **hosts.equiv** file only for client applications that do not supply a user account or password. On UNIX, the

hosts.equiv file is in the **/etc** directory. On Windows, the **hosts.equiv** file is in the **\%WINDIR%\system32\drivers\etc** directory. If you do not have a **hosts.equiv** file, you must create one.

On some networks, the host name that a remote host uses to connect to a particular computer might not be the same as the host name that the computer uses to refer to itself. For example, the network host name might contain the fully qualified domain name (FQDN), as in the following example:

```
fully_qualified_domain_name.informix.com
```

By contrast, the computer might refer to itself with the local host name, as the following example shows:

```
viking
```

If this situation occurs, make sure that you specify both host name formats in the **host.equiv** file.

To determine whether a client is trusted, execute the following statement on the client computer:

```
rlogin hostname
```

If you log in successfully without receiving a password prompt, the client is a trusted computer.

As an alternative, an individual user can list hosts from which he or she can connect as a trusted user in the **.rhosts** file. This file resides in the user's home directory on the computer on which the database server resides.

Windows Only:

On Windows, a home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the User Manager application.

You can increase security for Enterprise Replication and high availability cluster (High-Availability Data Replication, remote standalone secondary servers, and shared disk secondary servers) connections by configuring a dedicated port in the INFORMIXSQLHOSTS file. See the *IBM Informix Security Guide* for more information.

The netrc Information

The **netrc** information is optional information that specifies identity data. A user who does not have authorization to access the database server or is not on a computer that is trusted by the database server can use this file to supply a name and password that are trusted. A user who has a different user account and password on a remote computer can also provide this information.

On UNIX, the **netrc** information resides in the **.netrc** file in the user's home directory. Use any standard text editor to prepare the **.netrc** file. Windows maintains the **netrc** information in the registry keys. Use the Host Information tab of **setnet32** to edit the **netrc** information.

If you do not explicitly provide the user password in an application for a remote server (that is, through the USER clause of the CONNECT statement or the user name and password prompts in DB–Access), the client application looks for the

user name and password in the **netrc** information. If the user has explicitly specified the password in the application, or if the database server is not remote, the **netrc** information is not consulted.

The database server uses the **netrc** information regardless of whether it uses the default authentication policy or a communications support module.

For information about the specific content of this file, see your operating system documentation.

Windows Only:

On Windows, a home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the User Manager application.

User Impersonation:

For certain client queries or operations, the database server must impersonate the client to run a process or program on behalf of the client. In order to impersonate the client, the database server must receive a password for each client connection. Clients can provide a user ID and password through the CONNECT statement or **netrc** information.

The following examples show how you can provide a password to impersonate a client.

File or Statement
Example

netrc information
machine trngpc3 login bruce password im4golf

CONNECT statement
CONNECT TO ol_trngpc3 USER bruce USING "im4golf"

The sqlhosts File and the SQLHOSTS Registry Key

Informix client/server connectivity information, the *sqlhosts information*, contains information that enables a client application to find and connect to any Informix database server on the network.

For a detailed description of the **sqlhosts** information, see "The sqlhosts Information" on page 3-16.

The sqlhosts File (UNIX)

On UNIX, the **sqlhosts** file resides, by default, in the $**INFORMIXDIR/etc** directory. As an alternative, you can set the **INFORMIXSQLHOSTS** environment variable to the full path name and filename of a file that contains the **sqlhosts** file information. Each computer that hosts a database server or a client must have an **sqlhosts** file.

Each entry (each line) in the **sqlhosts** file contains the **sqlhosts** information for one database server. Use *white space* (spaces, tabs, or both) to separate the fields. Do not include any spaces or tabs *within* a field. To put comments in the **sqlhosts** file, start a line with the comment character (#). You can also leave lines completely blank for readability. Additional syntax rules for each of the fields are provided in the

following sections, which describe the entries in the **sqlhosts** file. Use any standard text editor to enter information in the **sqlhosts** file.

Table 3-2 shows a sample **sqlhosts** file.

Table 3-2. Sample sqlhosts File

dbservername	nettype	hostname	servicename	options
menlo	onipcshm	valley	menlo	
newyork	ontlitcp	hill	dynsrvr2	s=2,b=5120
sales	ontlispx	knight	sales	k=0,r=0
payroll	onsoctcp	dewar	py1	
asia	group	–	–	e=asia.3
asia.1	ontlitcp	node6	svc8	g=asia
asia.2	onsoctcp	node0	svc1	g=asia
portland	drsocssl	dewar	portland_serv	

Tools for Updating SQLHOSTS Information

To manage the SQLHOSTS information, use one of the following tools:

- Text editor
- IBM Informix Server Administrator (ISA)
- **setnet32**

Tip: Use ISA to manage the SQLHOSTS connectivity information. Although **setnet32** allows you to set up database servers (nettype, hostname, servicename, and options), it does not allow you to set up database server groups.

The SQLHOSTS Registry Key (Windows)

When you install the database server, the **setup** program creates the following key in the Windows registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS
```

This branch of the HKEY_LOCAL_MACHINE subtree stores the **sqlhosts** information. Each key on the SQLHOSTS branch is the name of a database server. When you click the database server name, the registry displays the values of the HOST, OPTIONS, PROTOCOL, and SERVICE fields for that particular database server.

Each computer that hosts a database server or a client must include the connectivity information either in the sqlhosts registry key or in a central registry. When the client application runs on the same computer as the database server, they share a single sqlhosts registry key.

Location of the SQLHOSTS Registry Key:

When you install the database server, the installation program asks where you want to store the SQLHOSTS registry key. You can specify one of the following two options:

- The local computer where you are installing the database server
- Another computer in the network that serves as a central, shared repository of **sqlhosts** information for multiple database servers in the network

Using a shared SQLHOSTS registry key relieves you of the necessity to maintain the same **sqlhosts** information on multiple computers. However, the **hosts** and **services** files on *each* computer must contain information about all computers that have database servers.

If you specify a shared sqlhosts registry key, you must set the **INFORMIXSQLHOSTS** environment variable on your local computer to the name of the Windows computer that stores the registry. The database server first looks for the sqlhosts registry key on the INFORMIXSQLHOSTS computer. If the database server does not find an sqlhosts registry key on the INFORMIXSQLHOSTS computer, or if **INFORMIXSQLHOSTS** is not set, the database server looks for an sqlhosts registry key on the local computer.

You must comply with Windows network-access conventions and file permissions to ensure that the local computer has access to the shared sqlhosts registry key. For information about network-access conventions and file permissions, see your Windows documentation.

Setting up SQLHOSTS with regedt32 (Windows):

Important: Use extreme caution with **regedt32**. If you make mistakes when editing the registry, you can destroy the configurations, not only of your IBM Informix products, but of your other applications.

To set up SQLHOSTS with regedt32 (Windows)

1. Run the Windows program, **regedt32**.
2. In the Registry Editor window, select the window for the HKEY_LOCAL_MACHINE subtree.
3. Click the folder icons to select the \SOFTWARE\INFORMIX\ SQLHOSTS branch.
4. With the SQLHOSTS key selected, add a new key.
5. Give the new key the name of the database server.
6. Select the new key that you just made (the key with the database server name) and add a new string value for it.
7. Give the value the name of one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, or SERVICE).

   Give the OPTIONS field the value of the database server group to which this database server will belong.
8. Modify the value to add value data.
9. Repeat steps 6 through 8 for each field of the **sqlhosts** information.

Sample SQLHOSTS Registry Key Information:

Figure 3-2 on page 3-16 illustrates the location and contents of the SQLHOSTS registry key for the database server **payroll**.

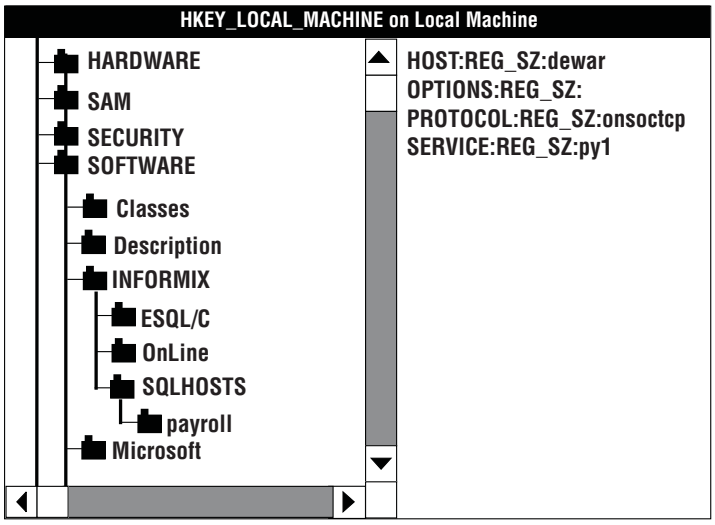


Figure 3-2. sqlhosts Information in the Windows Registry

The sqlhosts Information

The **sqlhosts** information in the **sqlhosts** file on UNIX or the SQLHOSTS registry key on Windows contains connectivity information for each database server. The **sqlhosts** information also contains definitions for groups. The database server looks up the connectivity information when you start the database server, when a client application connects to a database server, or when a database server connects to another database server.

The connectivity information for each database server includes four fields of required information and one optional field. The group information contains information in only three of its fields.

The five fields of connectivity information form one line in the UNIX **sqlhosts** file. On Windows, the database server name is assigned to a key in the SQLHOSTS registry key, and the other fields are values of that key. The following table summarizes the fields used for the **sqlhosts** information.

UNIX Field Name	Windows Field Name	Description of Connectivity Information	Description of Group Information
dbservername	Database server name key *or* database server group key	Database server name	Database server group name
nettype	PROTOCOL	Connection type	The word *group*
hostname	HOST	Host computer for the database server	*No information*. Use a hyphen as a placeholder in this field.
servicename	SERVICE	Alias for the port number	*No information*. Use a hyphen as a placeholder in this field.
options	OPTIONS	Options that describe or limit the connection	Group options

UNIX Only:

If you install IBM Informix Enterprise Gateway with DRDA in the same directory as the database server, your **sqlhosts** file also contains entries for

the Gateway and non-Informix database servers. However, this manual covers only the entries for the database server. For information about other entries in the **sqlhosts** file, see the *IBM Informix Enterprise Gateway with DRDA User Manual*.

IANA Standard Service Names and Port Numbers in the sqlhosts.std File

IANA, the Internet Assigned Numbers Authority, has assigned the following service names and port numbers for IBM Informix database servers:

Port/Service	IANA Code	Description
sqlexec	9088/tcp	IBM Informix SQL Interface
sqlexec-ssl	9089/tcp	IBM Informix SQL Interface - Encrypted

These service names now appear in the **sqlhosts.std** file of Dynamic Server. You do not need to change installed Informix Dynamic Server systems, because they will continue to work with existing port numbers and service names. (Also, there is no guarantee that some other system is not already using the service names or port numbers assigned to Informix Dynamic Server.)

Organizations that have policies for following standards can use these service names and port numbers if they want the database server to be in compliance with the IANA standard. If another application that is installed on the same machine already uses one of the service names or port numbers, you can ask the publisher of the non-compliant application to register for an IANA port number assignment to avoid the conflict. Note that while the noncompliant application remains noncompliant, you can run Dynamic Server using non-standard ports.

For more information, see the IANA organization web site.

Connectivity Information in Each sqlhosts Field

The next section describes the connectivity information that is in each field of the **sqlhosts** file or SQLHOSTS registry key.

Database Server Name

The database server name field (**dbservername**) gives the name of the database server for which the connectivity information is being specified. Each database server across all of your associated networks must have a unique database server name. The **dbservername** field must match the name of a database server in the network, as specified by the DBSERVERNAME and DBSERVERALIASES configuration parameters in the ONCONFIG configuration file. For more information about these configuration parameters, see “ONCONFIG Parameters for Connectivity” on page 3-32.

The **dbservername** field can include any printable character other than an uppercase character, a field delimiter, a new line character, or a comment character. This field is limited to 128 bytes.

UNIX Only:

If the **sqlhosts** file has multiple entries with the same dbservername, only the first one is used.

The Connection Type Field

The connection-type field (**nettype** on UNIX or PROTOCOL on Windows) describes the type of connection that should be made between the database server and the client application or another database server. The field is a series of eight letters composed of three subfields, as Figure 3-3 shows.

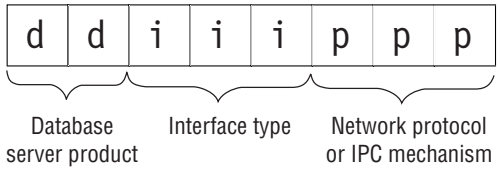


Figure 3-3. Format of the Connection-Type Field

The following sections describe the subfields of the connection-type field.

Database Server Product:

The first two letters of the connection-type field represent the database server product.

Product Subfield
Product

on *or* **ol**
The database server

dr (UNIX)
IBM Informix Enterprise Gateway with DRDA

For information about DRDA, see the *IBM Informix Enterprise Gateway with DRDA User Manual*.

For information on using the DRDA protocol to enable communication between Dynamic Server and IBM data server clients, see "Distributed Relational Database Architecture (DRDA) Communications" on page 3-36.

Interface Type:

The middle three letters of the connection-type field represent the network programming interface that enables communications. For more information, see "Network Programming Interface" on page 3-2.

Interface Subfield
Type of Interface

ipc IPC (interprocess communications)

soc Sockets

tli TLI (transport layer interface)

Interprocess communications (IPC) are used only for communications between two processes running on the same computer.

Network Protocol Entry:

The final three letters of the connection-type field represent the network protocol or specific IPC mechanism.

Protocol Subfield	Type of Protocol
imc	TCP/IP network protocol used with IBM Informix MaxConnect
nmp	Named-pipe communication
shm	Shared-memory communication
spx	IPX/SPX network protocol
str	Stream-pipe communication
tcp	TCP/IP network protocol

IPC connections use shared memory or stream pipes. The database server supports two network protocols: TCP/IP and IPX/SPX.

Table 3-3 summarizes the possible connection-type values for database server connections.

Table 3-3. Summary of nettype and Protocol Values

nettype Value (UNIX or Linux)	Protocol Value (Windows)	Description	Connection Type
drsocssl	drsocssl	Secured Sockets Layer (SSL) protocol for DRDA	Network
drsoctcp	drsoctcp	Distributed Relational Database Architecture™ (DRDA)	Network
drtlitcp	drtlitcp	DRDA	Network
onipcshm		Shared-memory communication	IPC
onipcstr		Stream-pipe communication	IPC
	onipcnmp	Named-pipe communication	IPC
ontlitcp		TLI with TCP/IP protocol	Network
onsocssl	onsocssl	SSL protocol	Network
onsoctcp	onsoctcp	Sockets with TCP/IP protocol	Network
ontlispx		TLI with IPX/SPX protocol	Network
onsocimc		Sockets with TCP/IP protocol for communication with Informix MaxConnect	Network
ontliimc		TLI with TCP/IP protocol for communication with Informix MaxConnect	Network
onsqlmux	onsqlmux	Single network connection with multiple database connections.	Network

For more information on Informix MaxConnect protocols, see "Using IBM Informix MaxConnect" on page 3-45.

To enable SQLMUX, use `sqlmux` as the **nettype** entry. For example, specify:

```
ids_mux  onsqlmux  ......
```

For information on the connection types for your platform, see "Connections That the Database Server Supports" on page 3-5.

If you are using the DRDA protocol to respond to requests from IBM data server clients, you must configure a new server alias in the SQLHOSTS file or Windows registry that uses either the **drtlitcp** and **drsoctcp** connection protocol.

For information on Connection Manager, see the instructions in Chapter 23, "Manage Cluster Connections with the Connection Manager," on page 23-1.

Host Name Field

The host name field (**hostname** on UNIX or host on Windows) contains the name of the computer where the database server resides. The name field can include any printable character other than a field delimiter, a new line character, or a comment character. The host name field is limited to 256 bytes.

The **hostname** field must be present, but is ignored if the connection type is `onsqlmux`.

The following sections explain how client applications derive the values used in the host name field.

Network Communication with TCP/IP:

When you use the TCP/IP network protocol, the host name field is a key to the **hosts** file, which provides the network address of the computer. The name that you use in the host name field must correspond to the *name* in the **hosts** file. In most cases, the host name in the **hosts** file is the same as the name of the computer. For information about the **hosts** file, see "TCP/IP Connectivity Files" on page 3-8.

In some situations, you might want to use the actual Internet IP address in the host name field. For information about using the IP address, see "IP Addresses for TCP/IP Connections" on page 3-28.

Shared-Memory and Stream-Pipe Communication (UNIX):

When you use shared memory or stream pipes for client/server communications, the **hostname** field must contain the actual host name of the computer on which the database server resides.

Network Communication with IPX/SPX (UNIX):

When you use the IPX/SPX network protocol, the **hostname** field must contain the name of the NetWare file server. The name of the NetWare file server is usually the UNIX *hostname* of the computer. However, this is not always the case. You might need to ask the NetWare administrator for the correct NetWare file server names.

Tip: NetWare installation and administration utilities might display the NetWare file server name in capital letters; for example, VALLEY. In the **sqlhosts** file, you can enter the name in either uppercase or lowercase letters.

Service Name Field

The interpretation of the service name field (**servicename** or service) depends on the type of connection that the connection-type field (**nettype** or PROTOCOL) specifies. The service name field can include any printable character other than a field delimiter, a new line character, or a comment character. The service name field is limited to 128 bytes.

The **servicename** field must be present, but is ignored if the connection type is `onsqlmux`.

Network Communication with TCP/IP:

When you use the TCP/IP connection protocol, the service name field must correspond to a service name entry in the **services** file. The port number in the **services** file tells the network software how to find the database server on the specified host. It does not matter what service name you choose, as long as you agree on a name with the network administrator.

Figure 3-4 shows the relationship between the **sqlhosts** file or registry and the **hosts** file, as well as the relationship of **sqlhosts** to the services file.

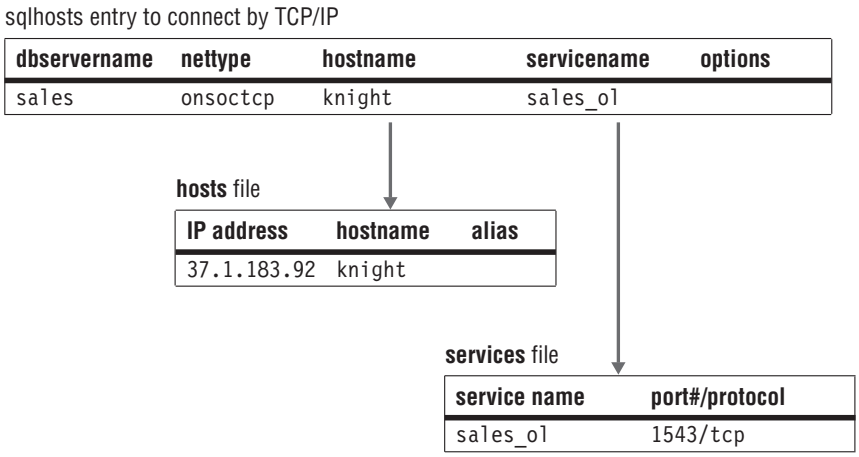


Figure 3-4. Relationship of sqlhosts File or Registry to hosts and services Files

In some cases, you might use the actual TCP listen-port number in the service name field. For information about using the port number, see "Port Numbers for TCP/IP Connections" on page 3-31.

Named-Pipe Communication (Windows):

When the PROTOCOL field specifies a named-pipe connection (**onipcnmp**), the SERVICE entry can be any short group of letters that is unique in the environment of the host computer where the database server resides.

Shared-Memory and Stream-Pipe Communication (UNIX):

When the **nettype** field specifies a shared-memory connection (`onipcshm`) or a stream-pipe connection (`onipcstr`), the database server uses the value in the **servicename** entry internally to create a file that supports the connection. For both `onipcshm` and `onipcstr` connections, the **servicename** can be any short group of letters that is unique in the environment of the host computer where the database server resides.

Recommendation: Use the **dbservername** as the **servicename** for stream-pipe connections.

Network Communication with IPX/SPX (UNIX):

A *service* on an IPX/SPX network is simply a program that is prepared to do work for you, such as the database server. For an IPX/SPX connection, the value in the **servicename** field can be an arbitrary string, but it must be unique among the names of services available on the IPX/SPX network. It is convenient to use the **dbservername** in the **servicename** field.

Options Field

The **options** field includes entries for the following features.

Option Name	Option Letter	Reference
Buffer size	b	page "Buffer-Size Option" on page 3-23
Connection redirection	c	page "Connection-Redirection Option" on page 3-23
End of group	e	page "End-of-Group Option" on page 3-24
Group	g	page "Group Option" on page 3-24
Identifier	i	page "Identifier Option" on page 3-27
Multiplexed connection	m	page "Multiplexed Connections" on page 3-4
Keep-alive	k	page "Keep-Alive Option" on page 3-27
Security	s (database server) r (client)	page "Security Options" on page 3-27
Communication support module	csm	*IBM Informix Security Guide*

When you change the values in the **options** field, those changes affect the next connection that a client application makes. You do not need to stop and restart the client application to allow the changes to take effect. However, a database server reads its own connectivity information *only* during initialization or database-server restart. If you change the options for the database server, you must restart the database server to allow the changes to take effect.

Syntax Rules for the Options Field:

Each item in the **options** field has the following format:

```
letter=value
```

You can combine several items in the **options** field, and you can include them in any order. The maximum length of the **options** field is 256 bytes.

You can use either a comma or white space as the separator between options. You cannot use white space within an option.

The database server evaluates the **options** field as a series of columns. A comma or white space in the **options** field represents an end of a column. Client and database server applications check each column to determine whether the option is supported. If an option is not supported, you are not notified. It is merely ignored.

The following examples show both valid and invalid syntax.

Syntax	Valid	Comments
k=0,s=3,b=5120	Yes	Syntax is correct.
s=3,k=0 b=5120	Yes	Syntax is equivalent to the preceding entry. (White space is used of instead of a comma.)
k=s=0	No	You cannot combine entries.

Buffer-Size Option: Use the buffer-size option (`b=`*value*) to specify in bytes the size of the communications buffer space. The buffer-size option applies only to connections that use the TCP/IP network protocol. Other types of connections ignore the buffer-size setting. You can use this option when the default size is not efficient for a particular application. The default buffer size for the database server using TCP/IP is 4096 bytes.

Adjusting the buffer size allows you to use system and network resources more efficiently; however, if the buffer size is set too high, the user receives a connection-reject error because no memory can be allocated. For example, if you set `b=64000` on a system that has 1000 users, the system might require 64 megabytes of memory for the communications buffers. This setting might exhaust the memory resources of the computer.

On many operating systems, the maximum buffer size supported for TCP/IP is 16 kilobytes. To determine the maximum allowable buffer size, see the documentation for your operating system or contact the technical-support services for the vendor of your platform.

If your network includes several different types of computers, be particularly careful when you change the size of the communications buffer.

Tip: Use the default size for the communications buffer. If you choose to set the buffer size to a different value, set the client-side communications buffer and the database server-side communications buffer to the same size.

Connection-Redirection Option: You can define a group of database servers, or aliases in the **sqlhosts** file on the client and have the client connections to the dbserver group try each of the group members with fail-over. The *connection-redirection* option (`c=`*value*) indicates the order in which the connection software chooses database servers, aliases, or coservers within a group.

Use the connection-redirection option in the following situations:

- To balance the load across multiple database server instances
- To use High-Availability Data Replication (HDR) to transfer over to a backup database server if a failure should occur

The following table shows the possible settings for the connection-redirection option.

Setting Result

c=0 By default, a client application connects to the first database server instance listed in the dbserver group in the **sqlhosts** file. If the client cannot connect to the first instance, it attempts to connect to the second instance and so on.

c=1 The client application chooses a random starting point from which to connect to a list of dbserver group members.

Important: The connection-redirection option is valid only in a dbserver group. For more information on the *g* option to specify dbserver groups, see "Group Option" on page 3-24.

Communication Support Module Option: Use the Communication Support Module (CSM) option to describe the CSM for each database server that uses a CSM. If you do not specify the CSM option, the database server uses the default authentication policy for that database server. You can specify the same CSM option setting for every database server described in the **sqlhosts** file or registry, or you can specify a different CSM option or no CSM options for each **sqlhosts** entry.

The format of the CSM option is illustrated in the following example:

```
csm=(csmname,csm-connection-options)
```

The value of *csmname* must match a *csmname* entry in the **concsm.cfg** file. The *connection-options* parameter overrides the default *csm-connection* options specified in the **concsm.cfg** file.

The following example specifies that the ENCCSM communication support module will be used for the connection:

```
csm=(ENCCSM)
```

For information on Communication Support Modules, see the *IBM Informix Security Guide*.

End-of-Group Option:

Use the end-of-group option (`e=`*dbservername*) to specify the ending database server name of a database server group. If you specify this option in an entry other than a database server group, the option is ignored.

If no end-of-group option is specified for a group, the group members are assumed to be contiguous. The end of group is determined when an entry is reached that does not belong to the group, or at the end of file, whichever comes first. For an example of the end-of-group option, see Table 3-4 on page 3-26.

Group Option:

When you define database server groups in the **sqlhosts** file or registry, you can use multiple related entries as one logical entity to establish or change client/server connections.

Important: Database server groups cannot be nested inside other database server groups, and database server group members cannot belong to more than one group.

Creating a Database Server Group in the sqlhosts File (UNIX):

To create a database server group in the sqlhosts file (UNIX)

1. Specify the name of the database server group to which the **sqlhosts** entry belongs (up to 128 bytes) in the DBSERVERNAME field.

   The database server group name can be the same as the initial DBSERVERNAME for the database server.
2. Place the keyword **group** in the connection type field.
3. The host name and service name fields are not used. Use dash (-) characters as null-field indicators for the unused fields. If you do not use options, you can omit the null-field indicators.

   The options for a database server group entry are as follows:

- c = connection redirection
- e = end of group
- g = group option
- i = identifier option

Setting Up the Database Server Group Registry Key Windows:

To set up the database server group registry key Windows

1. With the SQLHOSTS key selected, choose to add a new key.
2. Give the new key the name of the database server group.

   This value must correspond to the OPTIONS value in the database server name key.
3. Select the key with the database server group name that you just created and add a new string value for it.
4. Give the value the name of one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, SERVICE).
5. Add a value for the field.

   For a database server group, the **sqlhosts** information fields should have the following values:

   ```
   HOST       -
   OPTIONS    i=unique-integer-value
   PROTOCOL   group
   SERVICE    -
   ```

   Each database server group must have an associated identifier value (**i=**) that is unique among all database servers in your environment. Enter a minus (-) for HOST and SERVICE to indicate that you are not assigning specific values to those fields.
6. Repeat steps 4 and 5 for the remaining fields of the **sqlhosts**
7. Select the database server group key and choose to add a key.
8. Give the new key the name of the database server.

   This value must correspond to the database server key, whose OPTIONS value was set to the database server group key.
9. If you are combining Enterprise Replication with HDR, create keys for primary and secondary HDR servers under the same database server group.
10. Exit from the Registry Editor.

Using Database Server Groups for Enterprise Replication:

To use database server groups for Enterprise Replication, use the i and g options in Enterprise Replication. All database servers participating in replication must be a member of a database server group. Each database server in the enterprise must have a unique identifier, which is the server group. Make sure that the **sqlhosts** file is set up properly on each database server participating in replication.

For more information, see preparing the replication environment in the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Using Database Server Groups for High-Availability Data Replication:

To use database server groups for High-Availability Data Replication (HDR), use the `c`, `e`, and `g` options in HDR. HDR requires two identical systems. For more information, see "Directing Clients with the Connectivity Information" on page 19-33.

Sample Groups in the sqlhosts File or Registry:

The example in Table 3-4 shows the following two groups: **asia** and **peru**. Group **asia** includes the following members:

- **asia.1**
- **asia.2**
- **asia.3**

Because group **asia** uses the end-of-group option (`e=asia.4`), the database server searches for group members until it reaches `e=asia.4`, so the group includes **usa.2**. Group members do not need to be in order or even directly under the group to which they belong. Therefore, `asia.3` is valid despite its position after the **peru** group definition. Because group **peru** does not use the end-of-group option, the database server continues to include all members until it reaches the next group definition or the next group member definition that contains a `g=server` entry that specifies a different group. In this example, the **peru** group includes the servers **peru.1**, **peru.2**, **peru.3**, and **usa.1**.

Table 3-4 shows examples of database server groups in the **sqlhosts** file.

Table 3-4. Database Server Groups in sqlhosts File or Registry

dbservername	nettype	hostname	servicename	options
asia	group	–	–	e=asia.4
asia.1	ontlitcp	node6	svc8	g=asia
asia.2	onsoctcp	node0	svc1	g=asia
asia.3	drsocssl	node0	svc2	
usa.2	ontlispx	node9	sv2	
asia.4	onsoctcp	node1	svc9	g=asia
peru	group	–	–	
peru.1	ontlitcp	node4	svc4	
peru.2	ontlitcp	node5	svc5	g=peru
peru.3	ontlitcp	node7	svc6	
usa.1	onsoctcp	37.1.183.92	sales_ol	k=1, s=0
asia.3	onsoctcp	node10	svc10	g=asia

Using a Group Name in the INFORMIXSERVER and DBPATH Environment Variables:

You can use the name of a database server group instead of the database server name in the following environment variables, or in the SQL CONNECT command:

- **INFORMIXSERVER**

  The value of **INFORMIXSERVER** for a client application can be the name of a database server group. However, you cannot use a database server group name as the value of **INFORMIXSERVER** for a database server or database server utility.
- **DBPATH**

DBPATH can contain the names of database server groups as dbservernames.

Identifier Option:

The identifier option (`i=number`) assigns an identifying number to a database server group. The identifier must be a positive numeric integer and must be unique within your network environment.

For more information on the use of the identifier option, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Keep-Alive Option:

The keep-alive option is a network option that TCP/IP uses. It does not affect other types of connections. If you do not include the keep-alive option in the **options** field, the keep-alive feature is enabled by default. To set the keep-alive option on the database server side only, the client side only, or on both sides, specify **k=1** in the fifth column of the **sqlhosts** file. For most cases, you should enable the keep-alive option.

The letter *k* identifies keep-alive entries in the **options** field, as follows:

```
k=0     Disable the keep-alive feature
k=1     Enable the keep-alive feature
```

When a connected client and server are not exchanging data and the keep-alive option is enabled, the network service checks the connection periodically. If the receiving end of the connection does not respond within the time specified by the parameters of your operating system, the network service immediately detects the broken connection and frees resources.

When the keep-alive option is disabled, the network service does not check periodically whether the connection is still active. If the opposite end of the connection terminates unexpectedly without any notification, as when a PC reboots, for example, the network service might never detect that the connection is broken.

Multiplexed-Connection Option:

See "Multiplexed Connections" on page 3-4 for information on multiplexed connections.

Security Options:

The security options let you control operating-system security-file lookups. The letter *s* identifies database server-side settings, and the letter *r* identifies client-side settings. You can set both options in the **options** field. A client ignores *s* settings, and the database server ignores *r* settings.

The following table shows the possible security option settings.

Setting Result

r=0 Disables **netrc** lookup from the client side (no password can be supplied).

r=1 Enables **netrc** lookup from the client side (default setting for the client side).

s=0 Disables both **hosts.equiv** and **rhosts** lookup from the database server side (only incoming connections with passwords are accepted).

s=1 Enables only the **hosts.equiv** lookup from the database server side.

s=2 Enables only the **rhosts** lookup from the database server side.

s=3 Enables both **hosts.equiv** and **rhosts** lookup on the database server side (default setting for the database server side).

s=4 Configures a database server to use Pluggable Authentication Modules (PAM). See the *IBM Informix Security Guide* for more information.

s=6 Configures Enterprise Replication and High Availability Connection Security. See the *IBM Informix Security Guide* for more information.

s=7 Configures a database server to support single sign-on (SSO). See the *IBM Informix Security Guide* for more information.

The security options let you control the way that a client (user) gains access to a database server. By default, an Informix database server uses the following information on the client computer to determine whether the client host computer is trusted:

- **hosts.equiv**
- **rhosts** information

With the security options, you can specifically enable or disable the use of either or both files.

For example, if you want to prevent end users from specifying trusted hosts in **rhosts**, you can set `s=1` in the **options** field of the **sqlhosts** file or SQLHOSTS registry key for the database server to disable the **rhosts** lookup.

Important: Do not disable the **hosts.equiv** lookup in database servers that are used in distributed database operations. That is, if you expect to use the database server in distributed processing, do not set `s=0` or `s=2.`

Group Information

The following section describes the fields of the **sqlhosts** file or registry for groups.

Database Server Group

A database server group allows you to treat multiple related database server entries as one logical entity to establish or change client/server connections. You can also use dbserver groups to simplify the redirection of connections to database servers. For more information on database server groups, see “Group Option” on page 3-24.

Alternatives for TCP/IP Connections

The following sections describe some ways to bypass port and IP address lookups for TCP/IP connections.

IP Addresses for TCP/IP Connections

For TCP/IP connections (both TLI and sockets), you can use the actual IP address in the **hostname** field instead of the host name or alias found in the **hosts** file. The IP address is composed of four integers between `0` and `255,` separated by periods.

Table 3-5 shows sample IP addresses and hosts from a **hosts** file.

Table 3-5. A Sample hosts File

IP Address	Host Name	Host Alias
555.12.12.12	smoke	
98.765.43.21	odyssey	
12.34.56.789	knight	sales

Using the IP address for knight from Table 3-5, the following two **sqlhosts** entries are equivalent:

```
sales   ontlitcp   12.34.56.789   sales_ol
sales   ontlitcp   knight         sales_ol
```

Using an IP address might speed up connection time in some circumstances. However, because computers are usually known by their host name, using IP addresses in the host name field makes it less convenient to identify the computer with which an entry is associated.

UNIX Only:

You can find the IP address in the net address field of the **hosts** file, or you can use the UNIX **arp** or **ypmatch** command.

Windows Only:

You can configure Windows to use either of the following mechanisms to resolve Internet Domain Addresses (*mymachine*.informix.com) to Internet Protocol addresses (149.8.73.14):

- Windows Internet Name Service
- Domain Name Server

Wildcard Addressing for TCP/IP Connections

You can use wildcard addressing in the host name field when *both* of the following conditions are met:

- You are using TCP/IP connections.
- The computer where the database server resides has multiple network-interface cards (for example, three ethernet cards).

If the preceding conditions are met, you can use an asterisk (*) as a *wildcard* in the host name field that the database server uses. When you enter a wildcard in the host name field, the database server can accept connections at any valid IP address on its host computer.

Each IP address is associated with a unique host name. When a computer has multiple network-interface cards (NICs), as in Figure 3-5 on page 3-30, the **hosts** file must have an entry for each interface card. For example, the **hosts** file for the **texas** computer might include these entries.

NIC	Internet IP Address	Host Name
Card 1	123.45.67.81	texas1
Card 2	123.45.67.82	texas2

You can use the wildcard (*) alone or as a prefix for a host name or IP address, as shown in Table 3-6.

If the client application and database server share connectivity information (the **sqlhosts** file or the SQLHOSTS registry key), you can specify both the wildcard and a host name or IP address in the **host name** field (for example, `*texas1` or `*123.45.67.81`). The client application ignores the wildcard and uses the host name (or IP address) to make the connection, and the database server uses the wildcard to accept a connection from any IP address.

The wildcard format allows the listen thread of the database server to wait for a client connection using the same service port number on each of the valid network-interface cards. However, waiting for connections at multiple IP addresses might require slightly more CPU time than waiting for connections with a specific host name or IP address.

Figure 3-5 shows a database server on a computer (**texas**) that has two network-interface cards. The two client sites use different network cards to communicate with the database server.

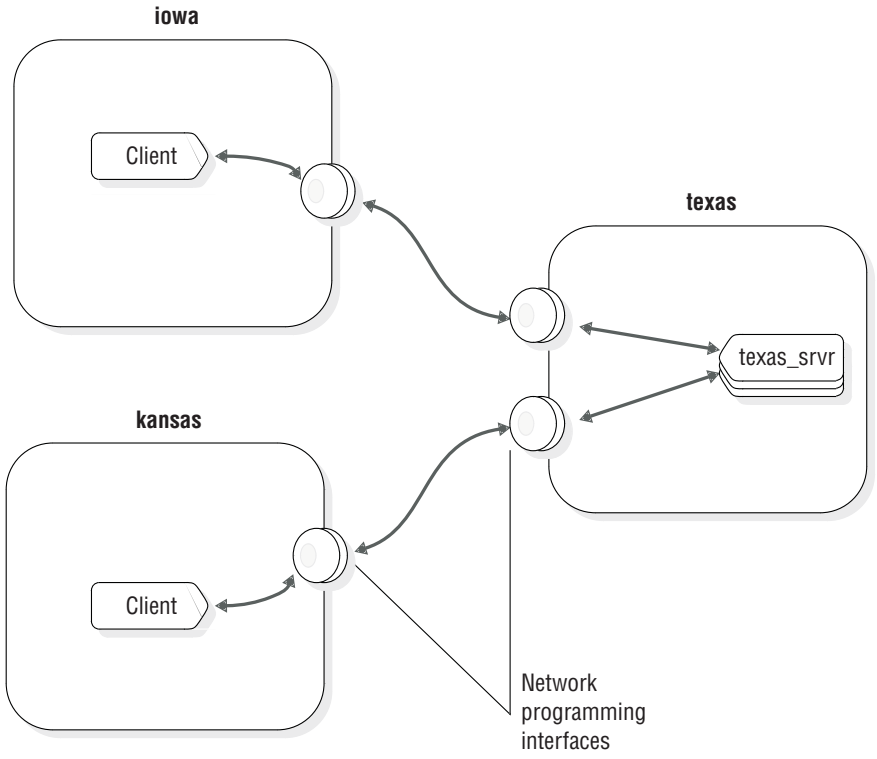


Figure 3-5. Using Multiple Network-Interface Cards

Table 3-6 shows the connectivity information for the **texas_srvr** database server.

Table 3-6. Possible Connectivity Entries for the texas_srvr Database Server

database server name	connection type	host name	service name
texas_srvr	ontlitcp	*texas1	pd1_on
texas_srvr	ontlitcp	*123.45.67.81	pd1_on
texas_srvr	ontlitcp	*texas2	pd1_on
texas_srvr	ontlitcp	*123.45.67.82	pd1_on

Table 3-6. Possible Connectivity Entries for the texas_srvr Database Server (continued)

database server name	connection type	host name	service name
texas_srvr	ontlitcp	*	pd1_on

Important: You can include only one of these entries.

If the connectivity information corresponds to any of the preceding lines, the **texas_srvr** database server can accept client connections from either of the network cards. The database server finds the wildcard in the **host name** field and ignores the explicit host name.

Tip: For clarity and ease of maintenance, you should include a host name when you use the wildcard in the host name field (that is, use `*host` instead of simply `*`).

The connectivity information used by a client application must contain an explicit host name or IP address. The client applications on **iowa** can use any of the following host names: `texas1`, `*texas1`, `123.45.67.81`, or `*123.45.67.81`. If there is a wildcard (*) in the **host name** field, the client application ignores it.

The client application on **kansas** can use any of the following host names: `texas2`, `*texas2`, `123.45.67.82`, or `*123.45.67.82`.

Port Numbers for TCP/IP Connections

For the TCP/IP network protocol, you can use the actual TCP listen port number in the service name field. The TCP port number is in the **port#** field of the **services** file.

For example, if the port number for the **sales** database server in the **services** file is `1543/tcp`, you can write an entry in the **sqlhosts** file as follows:

servername	nettype	hostname	servicename
sales	ontlitcp	knight	1543

Using the actual port number might save time when you make a connection in some circumstances. However, as with the IP address in the **host name** field, using the actual port number might make administration of the connectivity information less convenient.

Dynamic Server Support for IPv6 Addresses

Dynamic Server recognizes Internet Protocol Version 6 (IPv6) addresses, which are 128-bits long, on many platforms.

On all platforms, Dynamic Server supports Internet Protocol Version 4 (IPv4) addresses, which are 32-bits long.

IBM Informix Dynamic Server Version 10.00.xC4 and later and Client SDK 2.90.xC4 initially check to determine whether IPv6 is supported in the underlying operating system. If IPv6 is supported, the IPv6 address is used. If the underlying operating system does not support IPv6, the IPv4 address is used. Dynamic Server and CSDK retrieve the IP address from the name service.

You can treat Dynamic Server that runs on a host with both IPv4 and IPv6 addresses the same way you treat a server running on a multi-homed host. You can configure Dynamic Server on a host with both IPv4 and IPv6 addresses in either of the following ways:

- Create aliases (using the DBSERVERALIASES configuration parameter) and assign an IPv6 address to one of them and an IPv4 address to the other.
- Instruct the Dynamic Server to listen on all the IP addresses configured on the host by using a wild-carded hostname in the SQLHOSTS file.

  For example:

  ```
  olserver1   oltlitcp   *myhost   olservice1
  ```

  Note the asterisk in front of the hostname. This is the same naming scheme that currently exists for the servers in a multi-homed host. Starting with Dynamic Server Version 10.0, the hostname entry in the SQLHOSTS file maps to an IPv6 address if the host has a configured IPv6 address. If the host does not have a configured IPv6 address, the hostname entry maps to an IPv4 address. The hostname with a preceding asterisk maps to a wildcard address.

Disabling IPv6 Support

Dynamic server also provides a way to disable IPv6 support when working in IPv4 environments.

To disable IPv6 support for all database instances and client applications:

- Create an empty file called **$INFORMIXDIR/etc/IFX_DISABLE_IPV6**.

The file should have read permission for user informix. The file is not read from or written to, and does not need to contain any data.

To disable IPv6 support for a single database instance or for a single client application:

- On the database server instance, or on the machine on which applications are run, create an environment variable named **IFX_DISABLE_IPV6** and set its value to **yes**, as in:

  ```
  IFX_DISABLE_IPV6=yes
  ```

ONCONFIG Parameters for Connectivity

When you restart the database server, the restart procedure uses parameter values from the ONCONFIG configuration file. The following configuration parameters in the ONCONFIG file are related to connectivity:

- DBSERVERNAME
- DBSERVERALIASES
- LIMITNUMSESSIONS
- NETTYPE
- HA_ALIAS

The following topics contain information on these configuration parameters.

DBSERVERNAME Configuration Parameter

The DBSERVERNAME configuration parameter specifies a name, called the *dbservername*, for the database server. For example, to assign the value `nyc_research` to dbservername, use the following line in the ONCONFIG configuration file:

```
DBSERVERNAME nyc_research
```

When a client application connects to a database server, it must specify a dbservername. The **sqlhosts** information that is associated with the specified dbservername describes the type of connection that should be made.

Client applications specify the name of the database server in one of the following places:

- In the **INFORMIXSERVER** environment variable
- In SQL statements such as CONNECT, DATABASE, CREATE TABLE, and ALTER TABLE, which let you specify a database environment
- In the **DBPATH** environment variable

Windows Only:

In Windows, the DBSERVERNAME cannot be changed in the configuration file, because the registry stores information about the database server instance under the DBSERVERNAME.

The DBSERVERNAME must specify either the dbservername or one of the dbserveraliases. The name must begin with a lowercase letter and can contain other lowercase letters, digits, and underscores. The name should not include uppercase characters, a field delimiter (space or tab), or a new line character. Other characters from the basic ASCII code set are not necessarily reliable. For example, a hyphen or minus sign can create problems and a colon might not work reliably. The @ character is reserved to separate the database from the server (as in dbase@server).

You can configure DBSERVERALIASES connections as SSL connections, and you can have a mix of SSL and non-SSL connections.

DBSERVERALIASES Configuration Parameter

The DBSERVERALIASES configuration parameter lets you assign additional dbservernames to the same database server. The maximum number of aliases is 32. Figure 3-6 shows entries in an ONCONFIG configuration file that assign three dbservernames to the same database server instance.

```
DBSERVERNAME          sockets_srvr
DBSERVERALIASES       ipx_srvr,shm_srvr
```

Figure 3-6. Example of DBSERVERNAME and DBSERVERALIASES Parameters

The **sqlhosts** entries associated with the dbservernames from Figure 3-6 could include those shown in Figure 3-7 on page 3-34. Because each dbservername has a corresponding entry in the **sqlhosts** file or SQLHOSTS registry key, you can associate multiple connection types with one database server.

```
shm_srvr        onipcshm    my_host           my_shm
sockets_srvr    onsoctcp    my_host           port1
ipx_srvr        ontlispx    nw_file_server    ipx_srvr
```

Figure 3-7. Three Entries in the sqlhosts File for One Database Server in UNIX format

Using the **sqlhosts** file shown in Figure 3-7, a client application uses the following statement to connect to the database server using shared-memory communication:

```
CONNECT TO '@shm_srvr'
```

A client application can initiate a TCP/IP sockets connection to the *same* database server using the following statement:

```
CONNECT TO '@sockets_srvr'
```

DBSERVERALIASES must begin with a lowercase letter and can contain other lowercase letters, digits, and underscores. DBSERVERALIASES should not include uppercase characters, a field delimiter (space or tab), or a new line character. Other characters from the basic ASCII code set are not necessarily reliable. For example, a hyphen or minus sign can create problems and a colon might not work reliably. The @ character is reserved to separate the database from the server (as in dbase@server).

In the examples above, the `@shm_srvr` statement connects to an unidentified database at that server; alternatively, you could connect to `dbase1@shm_srvr`.

You can configure DBSERVERALIASES connections as SSL connections, and you can have a mix of SSL and non-SSL connections.

LIMITNUMSESSIONS Configuration Parameter

The LIMITNUMSESSIONS configuration parameter is an optional parameter that specifies the maximum number of sessions that you want connected to Dynamic Server. If you specify a maximum number, you can also specify whether you want Dynamic Server to print messages to the **online.log** file when the number of sessions approaches the maximum number.

Distributed queries against a server are counted against the limit.

You might need to dynamically increase or temporarily turn off the LIMITNUMSESSIONS configuration parameter to allow administrative utilities to run if the database server is reaching the limit. Use **onmode -wf** or **onmode -wm** to dynamically increase or turn off LIMITNUMSESSIONS.

If the LIMITNUMSESSIONS configuration parameter is enabled and sessions are restricted because of this limit, DBSA users can still initiate new sessions.

The LIMITNUMSESSIONS configuration parameter is not intended to be used as a means to adhere to license agreements.

Example

The following example specifies that you want a maximum of 100 sessions to connect to the database server and you want to print a warning message when the number of connected sessions approaches 100:

```
LIMITNUMSESSIONS 100,1
```

NETTYPE Configuration Parameter

The NETTYPE configuration parameter lets you adjust the number and type of virtual processors the database server uses for communication. Each type of network connection (ipcshm, ipcstr, ipcnmp, soctcp, tlitcp, and tlispx) can have a separate NETTYPE entry in the configuration file.

Recommendation: Although the NETTYPE parameter is not a required parameter, you should set NETTYPE if you use two or more connection types. After the database server has been running for some time, you can use the NETTYPE configuration parameter to tune the database server for better performance.

For more information about NETTYPE, see "Network Virtual Processors" on page 5-21. For information about the NETTYPE configuration parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

HA_ALIAS Configuration Parameter

The HA_ALIAS configuration parameter is an optional parameter used to define the name by which a secondary server is known within a high-availability cluster. In the event that the primary server in a high-availability environment fails, configure the Connection Manager to fail over to a secondary server identified by the HA_ALIAS parameter. See Chapter 23, "Manage Cluster Connections with the Connection Manager," on page 23-1 for information on configuring the arbitrator.

The value specified for HA_ALIAS must be one of the values already configured in DBSERVERNAME or DBSERVERALIASES. The value must also refer to a server whose connection type is a network protocol. The HA_ALIAS configuration parameter only affects RS Secondary and SD Secondary server types. If HA_ALIAS is not specified DBSERVERNAME is used.

Environment Variables for Network Connections

The **INFORMIXCONTIME** (connect time) and **INFORMIXCONRETRY** (connect retry) environment variables are *client* environment variables that affect the behavior of the client when it is trying to connect to a database server. Use these environment variables to minimize connection errors caused by busy network traffic.

If the client application explicitly attaches to shared-memory segments, you might need to set **INFORMIXSHMBASE** (shared-memory base). For more information, see "How a Client Attaches to the Communications Portion (UNIX)" on page 7-5.

The **INFORMIXSERVER** environment variable allows you to specify a default dbservername to which your clients will connect.

For more information on environment variables, see the *IBM Informix Guide to SQL: Reference*.

Distributed Relational Database Architecture (DRDA) Communications

This section contains information on how to configure Dynamic Server to use the Distributed Relational Database Architecture (DRDA), which is a set of protocols that permits multiple database systems, as well as application programs, to work together.

This section contains the following topics on the DRDA protocol:

Overview of DRDA

Distributed Relational Database Architecture (DRDA) is a set of protocols that enable communication between applications and database systems on disparate platforms and enables relational data to be distributed among multiple platforms. Any combination of relational database management products that use DRDA can be connected to form a distributed relational database management system. DRDA coordinates communication between systems by defining what must be exchanged and how it must be exchanged.

You can configure IBM Informix Dynamic Server to use DRDA to respond to requests from a common API (for example, from the IBM Data Server JDBC Driver and the IBM Data Server .NET Provider).

You can also enable DRDA connections between a client API and the Dynamic Server Connection Manager (the **oncmsm** utility), which manages and redirects client connection requests between an HDR primary server and HDR secondary servers. The Connection Manager provides automatic failover and load balancing capabilities. When a client makes a request to connect to a server, the Connection Manager routes the connection request to the server with the least CPU utilization. You can configure Connection Manager to automatically fail over to specific servers or to specific types of servers. For more information about the Connection Manager and instructions for configuring it, see Chapter 23, "Manage Cluster Connections with the Connection Manager," on page 23-1.

Enterprise Replication (ER), data replication, and Dynamic Server utilities, such as DB-Access, need SQLI connections. ER the utilities do not operate over DRDA connections. However, ER can coexist with DRDA.

Communication between a common API and Dynamic Server can use encrypted password security or an encrypted user ID and encrypted password security.

You can use the Secure Sockets Layer (SSL) protocol to encrypt data in end-to-end, secure TCP/IP and DRDA connections between Dynamic Server and a client common API. For information on SSL, see the *IBM Informix Security Guide*.

For more information on DRDA communications, see:

- "Configuring Dynamic Server for Connections to IBM Data Server Clients"
- "Allocating Poll Threads for an Interface/Protocol Combination With the NETTYPE Configuration Parameter" on page 3-38
- "Specifying the Size of the DRDA Communication Buffer With the DRDA_COMMBUFFSIZE Configuration Parameter" on page 3-38
- "The DRDAEXEC Thread and Queries from Clients" on page 3-39
- "SQL and Supported and Unsupported Data Types" on page 3-39
- "Displaying DRDA Connection Information" on page 3-40
- "Displaying DRDA Session Information" on page 3-40

Configuring Dynamic Server for Connections to IBM Data Server Clients

This topic explains how to configure Dynamic Server for connections to IBM data server clients.

Prerequisites: You must have installed an IBM Data Server Client and the applicable driver.

To configure Dynamic Server to connect to an IBM Data Server Client

1. Configure a new server alias in the SQLHOSTS file or Windows registry, using either the **drtlitcp** or **drsoctcp** connection protocol. Specify information in one of the following formats:

   ```
   server_name  drtlitcp machine_name
   service name/portnumber
   ```

   ```
   server_name  drsoctcp machine_name
   service name/portnumber
   ```

   For example, for a new SQLHOSTS file entry for a DRDA connection, specify:

   ```
   valley01_dr drtlitcp pdwest 9502
   ```

   If you are defining SQLHOSTS entries for the Connection Manager, include entries for the **drtlitcp** or **drsoctcp** connection protocol, as shown in the following list:

   ```
   ids_primary  onsoctcp  ids host ids port
   ids_hdr  onsoctcp  hdr host hdr port
   ids_sds  onsoctcp  sds host sds port
   ids_dr  drsoctcp  ids host ids drda port
   ids_hdr_dr drsoctcp  hdr host hdr drda port
   ids_sds_dr drsoctcp  sds host sds drda port
   ```

   For an example of Connection Manager entries, see "Connection Manager Setup Example" on page 23-5.

   For more information about SQLHOSTS entries, see "The sqlhosts File and the SQLHOSTS Registry Key" on page 3-13.
2. Verify that the ONCONFIG file lists the DRDA connection as one of the server aliases. The alias should not be the DBSERVERNAME.
3. Use the DRDA to connect to the server.

If you receive error 23104 when accessing the server through the DRDA protocol, the client application might be trying to bind a value that has a different encoding than the database locale. Set the environment variable GL_USEGLU to 1 before

you start the Dynamic Server. This enables the server to initialize the appropriate converters required to handle the code set conversions.

Also see:

- "Allocating Poll Threads for an Interface/Protocol Combination With the NETTYPE Configuration Parameter"
- "Specifying the Size of the DRDA Communication Buffer With the DRDA_COMMBUFFSIZE Configuration Parameter"
- "The DRDAEXEC Thread and Queries from Clients" on page 3-39
- "SQL and Supported and Unsupported Data Types" on page 3-39
- "Displaying DRDA Connection Information" on page 3-40
- "Displaying DRDA Session Information" on page 3-40

Allocating Poll Threads for an Interface/Protocol Combination With the NETTYPE Configuration Parameter

The NETTYPE configuration parameter configures poll threads for each connection type that your instance of the database server supports. You can use this configuration parameter to allocate more than one poll thread for an interface/protocol combination.

Set the NETTYPE configuration parameter as follows:

1. Specify **SQLI**, **drtlitcp**, or **drsoctcp** as the connection protocol.
2. Add information on the number of poll threads, the number of connections, and the virtual processor class.

For example, specify:

```
NETTYPE drtlitcp,3,2,CPU
```

A NETTYPE entry can handle multiple database server aliases on the same protocol type. Thus, when DRDA is in use, the network listener thread (NETTYPE **drtlitcp** or **drsoctcp**) typically has at least two sockets open and listening for connections. One socket is open for SQLI connections and another is open for DRDA connections. Additional sockets could be open if many separate server aliases are configured.

For more information on the NETTYPE configuration parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

Specifying the Size of the DRDA Communication Buffer With the DRDA_COMMBUFFSIZE Configuration Parameter

Use the DRDA_COMMBUFFSIZE configuration parameter to specify the size of the DRDA communications buffer. The minimum size is 4 kilobytes, the maximum size is 2 megabytes, and the default value is 32 kilobytes.

You can specify a one megabyte buffer as `1M`, `1m`, `1024K`, `1024k`, or `1024`. Dynamic Server automatically resets values that are less than 4 kilobytes as 32 kilobytes.

When a DRDA session is established, the session allocates a communication buffer of the current buffer size.

You can use the **isgetdrdacommbuffsize()** function to return the current value of DRDA_COMMBUFFSIZE.

You cannot use the **onmode -wm** command to change the setting while the database server is running.

The DRDAEXEC Thread and Queries from Clients

For every DRDA client, Dynamic Server creates a session and a DRDAEXEC thread, which is the equivalent of an SQLEXEC thread, to process and run the queries. This thread also formats the results of the queries in the DRDA protocol format and sends the results back to the client computer.

Queries issued from a DRDA client run in parallel if PDQPRIORITY is set and the query can run in parallel. Queries executed from DRDAEXEC threads can also run in parallel.

SQL and Supported and Unsupported Data Types

When using DRDA, Dynamic Server syntax is supported over the common API.

The following data types are supported over the common API:

- BIGINT
- BIGSERIAL
- BLOB
- BOOLEAN
- BYTE
- CHAR(32k)
- CLOB
- DATE
- DATETIME
- DECIMAL
- FLOAT
- INT
- INT8
- INTERVAL
- LVARCHAR(32k)
- MONEY
- NCHAR(32k)
- NVARCHAR(255)
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- TEXT
- VARCHAR(255)

When using DRDA connections, Dynamic Server rounds decimal and money values to 32-digit precision for all data retrieval operations on decimal or money data types.

Dynamic Server DATETIME values are mapped to DATE, TIME, or TIMESTAMP values.

The following data types are supported for use with database server host variables:

- CHAR
- DATE
- INT
- SMALLINT
- VCHAR

Displaying DRDA Connection Information

Use the following **onstat** and **onmode** commands to display information that includes the DRDA thread name and an indicator that distinguishes SQLI and DRDA sessions:

- **onstat -g ses**
- **onstat -g sql**
- **onstat -g ath**
- **onstat -g stk**
- **onstat -u**
- **onstat -x**
- **onstat -G**
- **onstat -g ddr**
- **onstat -g env**
- **onstat -g stm**
- **onstat -g ssc**
- **onmode -D**
- **onmode -Z**

For example, the **onstat** output might show "drdaexec" as the threadname.

Displaying DRDA Session Information

Use the **syssesappinfo** table in the **sysmaster** database to view DRDA client session information. The table shows the client session ID, session application name, and a session value in the **sesapp_sid**, **sesapp_name**, and **sesapp_value** columns.

For example, the table might show the following information:

- **sesapp_sid**: `6`
- **sesapp_name**: `Accting`
- **sesapp_value**: `db2jcc_application`

You can also display client session information using the **onstat -g ses** command.

Examples of Client/Server Configurations

The next several sections of this chapter show the correct entries in the **sqlhosts** file or SQLHOSTS registry key for several client/server connections. The following examples are included:

- Using a shared-memory connection
- Using a local-loopback connection
- Using a network connection
- Using multiple connection types
- Accessing multiple database servers

Important: In the following examples, you can assume that the network-configuration files hosts and services have been correctly prepared even if they are not explicitly mentioned.

Using a Shared-Memory Connection (UNIX)

Figure 3-8 shows a shared-memory connection on the computer named **river**.

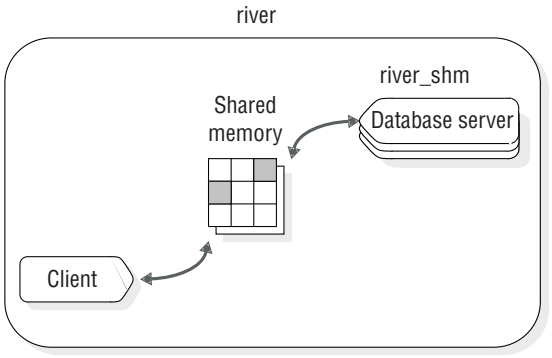


Figure 3-8. A Shared-Memory Connection

The ONCONFIG configuration file for this installation includes the following line:

```
DBSERVERNAME river_shm
```

The table below shows a sample entry in the **sqlhosts** file or SQLHOSTS registry key for the connection shown above in Figure 3-8.

Table 3-7. sqlhosts entry

dbservername	nettype	hostname	servicename
river_shm	onipcshm	river	rivershm

The client application connects to this database server using the following statement:

```
CONNECT TO '@river_shm'
```

Because this is a shared-memory connection, no entries in network configuration files are required. For a shared-memory connection, you can choose arbitrary values for the **hostname** and **servicename** fields of the **sqlhosts** file or SQLHOSTS registry key.

For more information about shared-memory connections, see "How a Client Attaches to the Communications Portion (UNIX)" on page 7-5.

Using a Local-Loopback Connection

Figure 3-9 shows a local-loopback connection that uses sockets and TCP/IP. The name of the host computer is **river**.

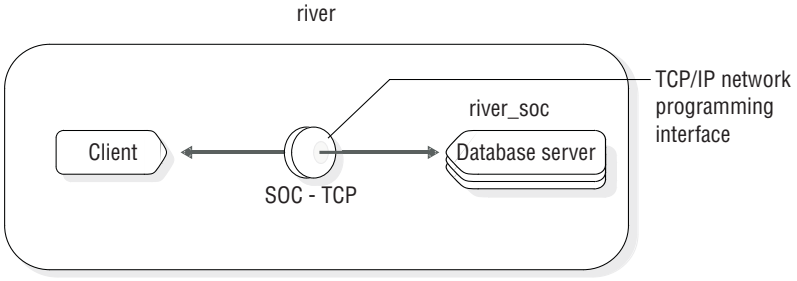


Figure 3-9. Local-Loopback Connection

The following table shows the correct entry for the **sqlhosts** file or SQLHOSTS registry key for the connection shown above in Figure 3-9.

Table 3-8. sqlhosts entry

dbservername	nettype	hostname	servicename
river_soc	onsoctcp	river	riverol

If the network connection uses TLI instead of sockets, only the **nettype** entry in this example changes. In that case, the **nettype** entry is `ontlitcp` instead of `onsoctcp`.

The **ONCONFIG** file includes the following line:

```
DBSERVERNAME river_soc
```

This example assumes that an entry for **river** is in the **hosts** file and an entry for **riverol** is in the **services** file.

Using a Network Connection

Figure 3-10 on page 3-43 shows a configuration in which the client application resides on host **river** and the database server resides on host **valley**.

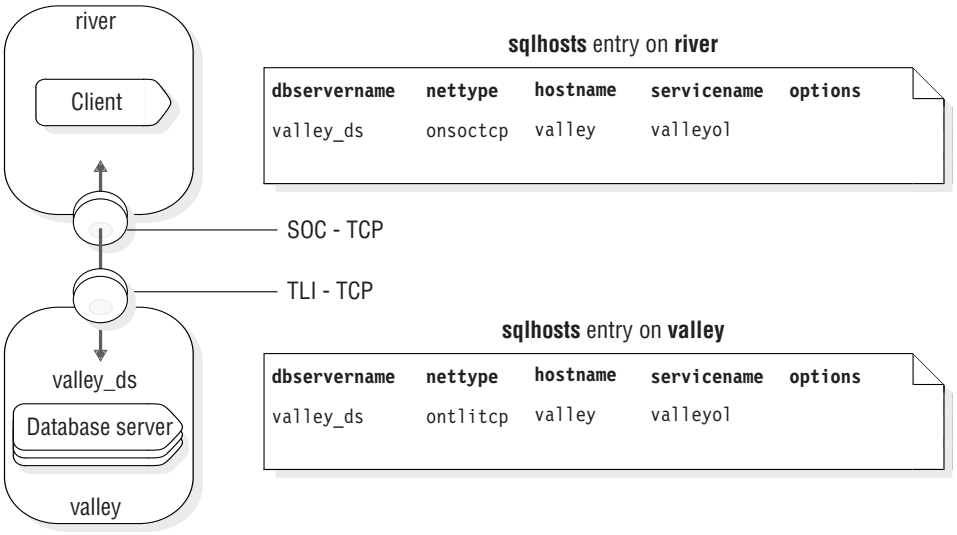


Figure 3-10. A Network Configuration

An entry for the **valley_ds** database server is in the **sqlhosts** files or registries on both computers. Each entry in the **sqlhosts** file or SQLHOSTS registry key on the computer where the database server resides has a corresponding entry on the computer where the client application resides.

UNIX Only:

Both computers are on the same TCP/IP network, but the host **river** uses sockets for its network programming interface, while the host **valley** uses TLI for its network programming interface. The **nettype** field must reflect the type of network programming interface used by the computer on which **sqlhosts** resides. In this example, the **nettype** field for the **valley_ds** database server on host **river** is `onsoctcp`, and the **nettype** field for the **valley_ds** database server on host **valley** is `ontlitcp`.

The sqlhosts File Entry for IPX/SPX (UNIX)

IPX/SPX software frequently provides TLI. Table 3-9 shows the entries in the **sqlhosts** file on both computers when the configuration in Figure 3-10 uses IPX/SPX instead of TCP/IP.

Table 3-9. sqlhosts entry

dbservername	nettype	hostname	servicename
valley_us	ontlispx	valley_nw	valley_us

In this case, the **hostname** field contains the name of the NetWare file server. The **servicename** field contains a name that is unique on the IPX/SPX network and is the same as the dbservername.

Using Multiple Connection Types

A single instance of the database server can provide more than one type of connection. Figure 3-11 on page 3-44 illustrates such a configuration. The database server is on host **river**. Client A connects to the database server with a shared-memory connection because shared memory is fast. Client B must use a network connection because the client and server are on different computers.

When you want the database server to accept more than one type of connection, you must take the following actions:

- Put DBSERVERNAME and DBSERVERALIASES entries in the ONCONFIG configuration file.
- Put an entry in the **sqlhosts** file or SQLHOSTS registry key for each database server/connection type pair.

For the configuration in Figure 3-11, the database server has two dbservernames: **river_net** and **river_shm**. The ONCONFIG configuration file includes the following entries:

```
DBSERVERNAME            river_net
DBSERVERALIASES         river_shm
```

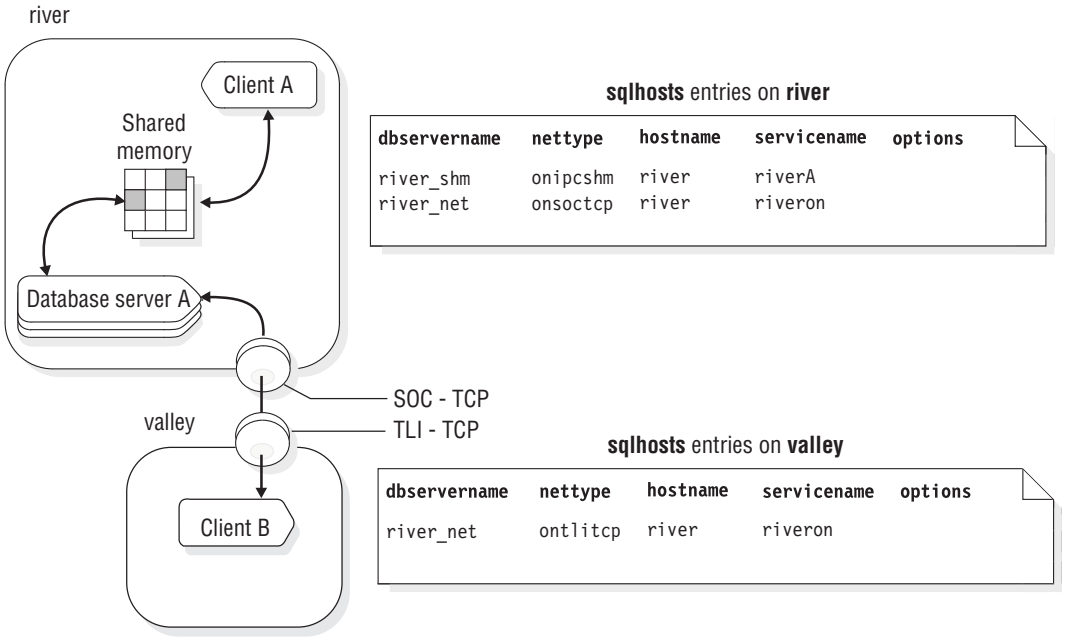


Figure 3-11. A UNIX Configuration That Uses Multiple Connection Types

The dbservername used by a client application determines the type of connection that is used. In Figure 3-11 above, Client A uses the following statement to connect to the database server:

```
CONNECT TO '@river_shm'
```

In the **sqlhosts** file or SQLHOSTS registry key, the **nettype** associated with the name **river_shm** specifies a shared-memory connection, so this connection is a shared-memory connection.

Client B uses the following statement to connect to the database server:

```
CONNECT TO '@river_net'
```

In the **sqlhosts** file or registry, the **nettype** value associated with **river_net** specifies a network (TCP/IP) connection, so client B uses a network connection.

Accessing Multiple Database Servers

Figure 3-12 shows a configuration with two database servers on host **river**. When more than one database server is active on one computer, it is known as *multiple residency*. (For more information about multiple residency, see your *IBM Informix Installation Guide*.)

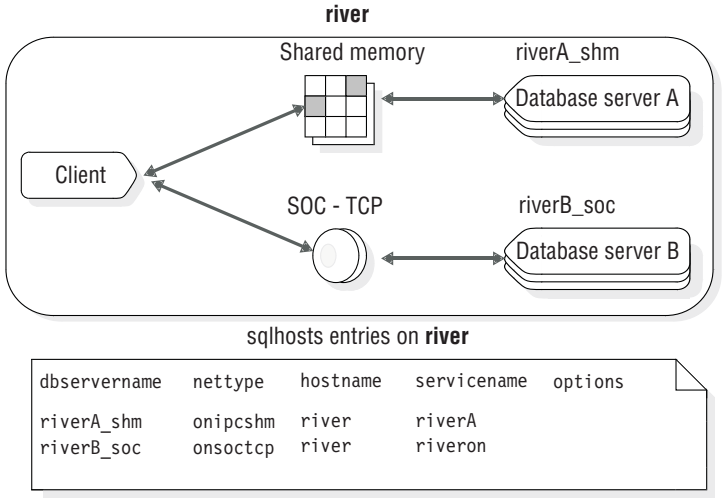


Figure 3-12. Multiple Database Servers on UNIX

For the configuration in Figure 3-12, you must prepare two ONCONFIG configuration files, one for database server **A** and the other for database server **B**. The **sqlhosts** file or SQLHOSTS registry key includes the connectivity information for both database servers.

The **ONCONFIG** configuration file for database server **A** includes the following line:

```
DBSERVERNAME            riverA_shm
```

The **ONCONFIG** configuration file for database server **B** includes the following line:

```
DBSERVERNAME riverB_soc
```

Using IBM Informix MaxConnect

IBM Informix MaxConnect is a networking product for Informix database server environments on UNIX. Informix MaxConnect manages large numbers (from several hundred to tens of thousands) of client/server connections. Informix MaxConnect multiplexes connections so that the ratio of client connections to database connections can be 200:1 or higher. Informix MaxConnect increases system scalability to many thousands of connections and saves system resources, reducing response times and CPU requirements. Informix MaxConnect is best for OLTP data transfers and not recommended for large multimedia data transfers.

Install Informix MaxConnect separately from your Informix database server and client applications. For maximum performance benefit, install Informix MaxConnect either on a separate computer to which Informix clients connect or on the client application server. You can install Informix MaxConnect in the following configurations:

- On a dedicated server to which Informix clients connect
- On the client application server
- On the database server computer

Two protocols for multiplexing connections, **ontliimc** and **onsocimc**, are available for Informix MaxConnect users. You can use the **ontliimc** and **onsocimc** protocols in the following two configurations:

- To connect Informix MaxConnect to the database server.

  In this configuration, the client connections are multiplexed and use packet aggregation.
- To connect the client applications directly to the database server without going through Informix MaxConnect.

  In this configuration, the client does not get the benefits of connection multiplexing or packet aggregation. Choose this configuration when the client application is transferring simple- or smart-large-object data, because a direct connection to the database server is best.

For more information on how to configure Informix MaxConnect and monitor it with the **onstat -g imc** and **imcadmin** commands, see the *IBM Informix MaxConnect User's Guide*.

Important: Informix MaxConnect and the *IBM Informix MaxConnect User's Guide* ship separately from the Informix database server.

Chapter 4. Initializing the Database Server

In This Chapter

This chapter tells how to initialize the database server, describes the activities that take place during initialization, describes database server operating modes, and provides information on changing operating modes.

Types of Initialization

Initialization of the database server refers to two related activities: shared-memory initialization and disk-space initialization.

Shared-memory initialization or bringing up or starting the server establishes the contents of database server shared memory as follows: internal tables, buffers, and the shared-memory communication area. Shared memory is initialized every time the database server starts up. You use the **oninit** utility from the command line to initialize database server shared memory and bring the database server online.

Shared-memory initialization also occurs when you restart the database server.

One key difference distinguishes shared-memory initialization from disk-space initialization:

- Shared-memory initialization has no effect on disk-space allocation or layout. No data is destroyed.

Disk-space initialization uses the values stored in the configuration file to create the initial chunk of the root dbspace on disk. When you initialize disk space, the database server automatically initializes shared memory as part of the process. Disk space is initialized the first time the database server starts up. It is only initialized thereafter during a cold restore or at the request of the database server administrator.

Warning: When you initialize disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing database server, all the data in the earlier database server becomes inaccessible and, in effect, is destroyed.

Initializing Disk Space

The database server must be in offline mode when you begin initialization.

If you are starting a database server for the first time or you want to remove all dbspaces and their associated data, use the following methods to initialize the disk space and to bring the database server into online mode.

Operating System	Action to Bring Database Server into Online Mode
UNIX	You must be logged in as **informix** or **root** to initialize the database server. Execute **oninit -iy**.

Operating System	Action to Bring Database Server into Online Mode
Windows	You must be a member of the **Administrators** or **Power Users** group to initialize the database server. • The database server runs as a service. In the **Services** control application, choose the database server service and type **-iy** in the Startup parameters field. Then click **Start**. • On the command line, use the **starts** *dbservername* **-iy** command.

Warning: When you execute these commands, all existing data in the database server disk space is destroyed. Use the **-i** flag only when you are starting a new instance of the database server.

You can use the **oninit -s** option to initialize shared memory and leave the database server in quiescent mode.

Windows Only:

When you install the database server and choose to initialize a new instance of the database server, or when you use the instance manager program to create a new instance of the database server, the database server is initialized for you.

Recommendation: Do not use the **oninit -iy** command to initialize the database server unless you are diagnosing a problem.

Automatic system startup is not available via SQL administration. You must start the database server according to information in this section. If you use the SQL administration API and the Scheduler, when you use **oninit** to start the database server, the server automatically executes any stored procedures scheduled to run when the server starts. For more information, see Part 6, "Automatic Monitoring and Corrective Actions."

For more information about **oninit**, see the *IBM Informix Dynamic Server Administrator's Reference*.

For information on using these utilities to initialize the database server and change the database server mode, see "Starting the Database Server and Initializing Disk Space" on page 1-12.

Initialization Steps

Disk-space initialization always includes the initialization of shared memory. However, some activities that normally take place during shared-memory initialization, such as recording configuration changes, are not required during disk initialization because those activities are not relevant with a newly initialized disk.

Table 4-1 shows the main tasks completed during the two types of initialization. The following sections discuss each step.

Table 4-1. Initialization Steps

Shared-Memory Initialization	Disk Initialization
Process configuration file.	Process configuration file.
Create shared-memory segments.	Create shared-memory segments.
Initialize shared-memory structures.	Initialize shared-memory structures.

Table 4-1. Initialization Steps (continued)

Shared-Memory Initialization	Disk Initialization
	Initialize disk space.
Start all required virtual processors.	Start all required virtual processors.
Make necessary conversions.	
Initiate fast recovery.	
Initiate a checkpoint.	Initiate a checkpoint.
Document configuration changes.	
Update **oncfg_servername.servernum** file.	Update **oncfg_servername.servernum** file.
Change to quiescent mode.	Change to quiescent mode.
Drop temporary tblspaces (optional).	
Set forced residency, if requested.	Set forced residency, if specified.
Change to online mode and return control to user.	Change to online mode and return control to user.
If the SMI tables are not current, update the tables.	Create **sysmaster** database that contains the SMI tables.
	Create the **sysutils** database.
	Create the **sysuser** database
	Create the **sysadmin** database
Monitor maximum number of user connections at each checkpoint.	Monitor maximum number of user connections at each checkpoint.

Process Configuration File

The database server uses configuration parameters to allocate shared-memory segments during initialization and restart. If you modify a shared-memory configuration parameter, you must shut down and restart the database server for the change to take effect.

The **ONCONFIG**, **onconfig**, and **onconfig.std** files are stored in **$INFORMIXDIR/etc** on UNIX and **%INFORMIXDIR%\etc** on Windows. During initialization, the database server looks for configuration values in the following files, in this order:

1. If the **ONCONFIG** environment variable is set, the database server reads values from the **ONCONFIG** file.

   If the **ONCONFIG** environment variable is set, but the database server cannot access the specified file, it returns an error message.
2. If the **ONCONFIG** environment variable is not set, the database server reads the configuration values from the **ONCONFIG** file.
3. If you omit a configuration parameter in your **onconfig** file, the database server reads the configuration values from the **$INFORMIXDIR/etc/onconfig.std** file.

You should *always* set the **ONCONFIG** environment variable before you initialize or restart the database server. The default configuration files are intended as templates and not as functional configurations. However, the server will not initialize if **onconfig_std** is missing. For more information about the configuration file, see "Configure the Database Server Configuration" on page 1-10.

The restart process compares the values in the current configuration file with the previous values, if any, that are stored in the root dbspace reserved page,

PAGE_CONFIG. When differences exist, the database server uses the values from the current ONCONFIG configuration file when the database server is restarted.

Create Shared-Memory Portions

The database server uses the configuration values to calculate the required size of the database server resident shared memory. In addition, the database server computes additional configuration requirements from internal values. Space requirements for overhead are calculated and stored.

To create shared memory, the database server acquires the shared-memory space from the operating system for three different types of memory:

- Resident portion, used for data buffers and internal tables
- Virtual portion, used for most system and user-session memory requirements
- IPC communication portion, used for IPC communication

  The database server allocates this portion of shared memory only if you configure an IPC shared-memory connection.

Next, the database server attaches shared-memory segments to its virtual address space and initializes shared-memory structures. For more information about shared-memory structures, see "Virtual Portion of Shared Memory" on page 7-13.

After initialization is complete and the database server is running, it can create additional shared-memory segments as needed. The database server creates segments in increments of the page size.

Initialize or Restart Shared-Memory

After the database server attaches to shared memory, it clears the shared-memory space of uninitialized data. Next the database server lays out the shared-memory header information and initializes data in the shared-memory structures. The database server lays out the space needed for the logical-log buffer, initializes the structures, and links together the three individual buffers that form the logical-log buffer. For more information about these structures, see the **onstat** utility section in the *IBM Informix Dynamic Server Administrator's Reference*.

After the database server remaps the shared-memory space, it registers the new starting addresses and sizes of each structure in the new shared-memory header.

During shared-memory initialization, disk structures and disk layout are not affected. The database server reads essential address information, such as the locations of the logical and physical logs, from disk and uses this information to update pointers in shared memory.

Initialize Disk Space

This procedure is performed only during disk-space initialization, not when the database server is restarted. After shared-memory structures are initialized, the database server begins initializing the disk. The database server initializes all the reserved pages that it maintains in the root dbspace on disk and writes PAGE_PZERO control information to the disk.

Start All Required Virtual Processors

The database server starts all the virtual processors that it needs. The parameters in the ONCONFIG file influence what processors are started. For example, the NETTYPE parameter can influence the number and type of processors started for making connections. For more information about virtual processors, see "Virtual Processors" on page 5-1.

Make Necessary Conversions

The database server checks its internal files. If the files are from an earlier version, it updates these files to the current format. For information about database conversion, see the *IBM Informix Migration Guide*.

Start Fast Recovery

The database server checks if fast recovery is needed and, if so, starts it. For more information about fast recovery, see "Fast Recovery" on page 15-7.

Fast recovery is not performed during disk-space initialization because there is not yet anything to recover.

Start a Checkpoint

After fast recovery completes, the database server executes a checkpoint to verify that all recovered transactions are flushed to disk so the transactions do not need to be repeated.

As part of the checkpoint procedure, the database server writes a checkpoint-complete message in the message log. For more information about checkpoints, see "Checkpoints" on page 15-4.

The database server now moves to quiescent mode or online mode, depending on how you started the initialization or database-server restart process.

Document Configuration Changes

The database server compares the current values stored in the configuration file with the values previously stored in the root dbspace reserved page PAGE_CONFIG. When differences exist, the database server notes both values (old and new) in a message to the message log.

This task is not performed during disk-space initialization or restart.

Create the oncfg_servername.servernum File

The database server creates the **oncfg_*servername.servernum*** file and updates it every time that you add or delete a dbspace, blobspace, logical-log file, or chunk. You do not need to manipulate this file in any way, but you can see it listed in your **$INFORMIXDIR/etc** directory on UNIX or in your %INFORMIXDIR%**\etc** directory on Windows. The database server uses the **oncfg_*servername.servernum*** file during a full-system restore for salvaging the logical log.

For more information about the **oncfg_*servername.servernum*** file, see the section on files that the database server uses in the *IBM Informix Dynamic Server Administrator's Reference*.

Drop Temporary Tblspaces

The database server searches through all dbspaces for temporary tblspaces. (If you use the **-p** option of **oninit** to initialize the database server, the database server skips this step.) These temporary tblspaces, if any, are tblspaces left by user processes that died prematurely and were unable to perform proper cleanup. The database server deletes any temporary tblspaces and reclaims the disk space. For more information about temporary tblspaces, see "Temporary Tables" on page 9-25.

This task is performed when the database server is restarted; it is not performed during disk-space initialization.

Set Forced Residency If Specified

If the value of the RESIDENT configuration parameter is `-1` or a number greater than 0, the database server tries to enforce residency of shared memory. If the host computer system does not support forced residency, the initialization procedure continues. Residency is not enforced, and the database server sends an error message to the message log. For more information about the RESIDENT configuration parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

Return Control to User

The database server writes the `IBM Informix Dynamic Server initialized - complete disk initialized` message in the message log only if initialization, not database-server restart, occurred. The database server also dynamically allocates a virtual shared-memory segment.

At this point, control returns to the user. Any error messages generated by the initialization procedure are displayed in the following locations:

- The command line
- The database server message log file, specified by the MSGPATH configuration parameter

  For more information about the MSGPATH parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.
- **Summary** section of IBM Informix Server Administrator (ISA)

You can use the **oninit -w** utility to force the server to return to a command prompt within a configurable timeout. The **oninit -w** utility is useful for troubleshooting initialization failures. For syntax and information about **oninit**, see the *IBM Informix Dynamic Server Administrator's Reference*.

Create sysmaster Database and Prepare SMI Tables

Even though the database server has returned control to the user, it has not finished its work. The database server now checks the *system-monitoring interface* (SMI) tables. If the SMI tables are not current, the database server updates the tables. If the SMI tables are not present, as is the case when the disk is initialized, the database server creates the tables. After the database server builds the SMI

tables, it puts the message `sysmaster database built successfully` in the message log. The database server also re-creates the **sysmaster** database during conversion and reversion. For more information about SMI tables, see the chapter on the **sysmaster** database in the *IBM Informix Dynamic Server Administrator's Reference*.

If you shut down the database server before it finishes building the SMI tables, the process of building the tables stops. This condition does not damage the database server. The database server simply builds the SMI tables the next time that you bring the database server online. However, if you do not allow the SMI tables to finish building, you cannot run any queries against those tables, and you cannot use ON–Bar for backups.

After the SMI tables have been created, the database server is ready for use. The database server runs until you stop it or, possibly, until a malfunction.

Recommendation: *Do not* try to stop the database server by stopping a virtual processor or ending another database server process. For more information, see "Starting and Stopping Virtual Processors" on page 6-3.

Create the sysutils Database

The database server drops and re-creates the **sysutils** database during disk initialization, conversion, or reversion. ON–Bar stores backup and restore information in the **sysutils** database. Wait until the message `sysutils database built successfully` displays in the message log. For more information, see the *IBM Informix Backup and Restore Guide*.

Create the sysuser Database

The **sysuser** database is used for PAM (Pluggable Authentication Module) authentication in IDS server to server communication.

Create the sysadmin Database

The **sysadmin** database provides remote administration and scheduler API features in IDS.

Monitor Maximum Number of User Connections

At each checkpoint, the database server prints the maximum number of user connections in the message log: `maximum server connections` *number*. You can monitor the number of users who have connected to the database server since the last restart or disk initialization.

The number displayed is reset when the customer reinitializes the database server.

Database Server Operating Modes

You can determine the current database server mode by executing the **onstat** utility from the command line. The **onstat** header displays the mode.

Table 4-2 on page 4-8 shows the principal modes of operation of the database server.

Table 4-2. Operating Modes

Operating Mode	Description	Users Allowed Access
Offline mode	The database server is not running. Shared memory is not allocated.	Only the administrator (user **informix**) can change from this mode to another mode.
Quiescent mode	Database-server processes are running and shared-memory resources are allocated. Administrators use this mode to perform maintenance functions that do not require the execution of SQL and DDL statements.	Only the administrator (user **informix**) can access the database server. Other users can view database-server status information, but they cannot access the database server.
Administration mode	This mode is an intermediary mode between Quiescent mode and Online mode. Administrators use this mode to perform any maintenance task, including tasks requiring the execution of SQL and DDL statements. Administrators can also perform all other functions available in Online mode.	The following users can connect to the database server in administration mode: • User **informix** • Users who have the DBSA role Set the ADMIN_USER_MODE_WITH_DBSA configuration parameter to 1 if you want users who are members of the DBSA group (in addition to user **informix**) to connect to the database server in administration mode. • One or more users who have administration mode access User **informix** or a DBSA can dynamically give one or more specific users the ability to connect to the database server in administration mode through the **onmode -j** command, the **oninit -U** command, or the ADMIN_MODE_USERS configuration parameter. Other users can view database-server status information, but they cannot access the database server.
Online mode	This is the normal operating mode of the database server.	Any authorized user can connect with the database server and perform all database activities. User **informix** or user **root** can use the command-line utilities to change many database server ONCONFIG parameter values.

In addition, the database server can also be in one of the following modes:

- *Read-only mode* is used by the secondary database server in a data replication environment. An application can query a secondary database server that is in read-only mode, but the application cannot write to a read-only database.
- *Recovery mode* is transitory. It occurs when the database server performs fast recovery or recovers from a system archive or system restore. Recovery occurs during the change from offline to quiescent mode.

- *Shutdown mode* is transitory. It occurs when the database server is moving from online to quiescent mode or from online (or quiescent) to offline mode. Current users access the system, but no new users are allowed access.

  After shutdown mode is initiated, it cannot be cancelled.

Changing Database Server Operating Modes

This section describes how to change from one database server operating mode to another with the **oninit** and **onmode** utilities and with ISA. It also contains information on using the ADMIN_MODE_USERS configuration parameter to specify which users can connect to the server in administration mode.

Windows Only:

In Windows, the database server runs as a service. Windows provides a service control application (also called the Services tool) to start, stop, and pause a service. The service control application is located in the Control Panel program group. The service name for the database server includes the database server name (the value of DBSERVERNAME in the ONCONFIG file). For example, the Dynamic Server service for the `newyork` database server is:

```
IBM Informix Database Server - newyork
```

To change mode with the Services tool, start the tool and select the database server service. Then choose the appropriate button in the Services window. The tables shown later in this chapter explain which button you select for each mode.

To start and stop the database server, you can use other Windows tools, such as the NET command and the Server Manager tool. For more information about these methods, consult your Windows operating-system documentation.

Tip: After you change the mode of your database server, execute the **onstat** command to verify the current server status.

Users Permitted to Change Modes

UNIX Only:

Users who are logged in as **root** or **informix** can change the operating mode of the database server. If the INF_ROLE_SEP environment variable is set, the database server administrator can also change the operating mode of the database server. The INF_ROLE_SEP environment variable enforces separating administrative tasks by people who run and audit the database server.

If you want users with the DBSA group to connect to the database server in administration mode, set the ADMIN_USER_MODE_WITH_DBSA configuration parameter to 1. If this parameter is set to zero, then access is restricted to user **informix** only.

User **informix** or a DBSA can dynamically give one or more specific users the ability to connect to the database server in administration mode, using the **onmode** utility, the **oninit** utility, or the ADMIN_MODE_USERS configuration parameter.

Windows Only:

Table 4-2 on page 4-8 shows which users can change the operating mode of the database server in Windows. If you use ISA, you can log in to the operating system as any user but you must log in to Apache as user **informix**. The Apache server runs as a member of the **Informix-Admin** group.

Table 4-3. Changing Operating Modes in Windows

Changing Operating Mode	Administrators Group	Informix-Admin Group
command-line utilities such as **starts**		X
ISA		X
Services control panel	X	

ISA Options for Changing Modes

You can use ISA to change the database server mode. For more information, see the ISA online help.

Action	ISA Option
Initialize the database server.	**Mode > Online (Initialize Disks)**or **Quiescent (Initialize Disks)**
Change from offline or administration to quiescent.	**Mode > Quiescent**
Change from any mode to administration.	**Mode > Single-User**
Change from offline, quiescent, or administration to online.	**Mode > Online**
Change gracefully from online to quiescent.	**Mode > Quiescent (Graceful)**
Change immediately from online to quiescent.	**Mode > Quiescent (Immediate)**
Shut down the database server.	**Mode > Offline**

ON-Monitor Options for Changing Modes (UNIX)

You can use ON–Monitor to change the database server mode on UNIX. For more information, see the section on ON–Monitor in the *IBM Informix Dynamic Server Administrator's Reference*.

Action	Menu Option
Initialize the database server.	**Parameters > Initialize**
Change from offline to quiescent.	**Mode > Startup**
Change from offline or quiescent to online.	**Mode > Online**
Change gracefully from online to quiescent.	**Mode > Graceful Shutdown**
Change immediately from online to quiescent.	**Mode > Immediate Shutdown**
Change to administration mode.	**Mode > Single User**
Shut down the database server.	**Mode > Take Offline**

Command-Line Options for Changing Modes

Table 4-2 on page 4-8 contains descriptions of each mode and shows which users can access the database server when the server is in each mode. This section,

which contains information on commands for changing modes and information about how mode changes effect user sessions, has the following subsections:

- "Change From Offline to Quiescent Mode"
- "Change From Offline to Online Mode"
- "Change From Offline to Administration Mode"
- "Change From Quiescent to Online Mode" on page 4-12
- "Change Gracefully from Online to Quiescent Mode" on page 4-12
- "Change Immediately from Online to Quiescent Mode" on page 4-13
- "Change From Quiescent or Online to Administration Mode" on page 4-13
- "Change From Administration to Online Mode" on page 4-14
- "Change From Administration to Quiescent Mode" on page 4-14
- "Change From Any Mode Immediately to Offline Mode" on page 4-14

Also see "Specifying Administration Mode Users with the ADMIN_MODE_USERS Configuration Parameter" on page 4-15.

Change From Offline to Quiescent Mode

When the database server changes from offline mode to quiescent mode, the database server initializes shared memory. Only administrators can access the database server to perform maintenance functions that do not involve the execution of SQL and DDL statements.

Operating System
Action

UNIX

- Execute **oninit -s**.

Windows

- On the command line, use the **starts dbservername -s** command.

Change From Offline to Online Mode

When you move the database server from offline to online mode, the database server initializes shared memory and is available for all user sessions.

Operating System
Action

UNIX

- Execute **oninit**.

Windows

- With the Services tool, select the database server service and click **Start.**
- On the command line, use the **starts** *dbservername* command.

Change From Offline to Administration Mode

When you move the database server from off-line to administration mode, you move the server into a mode that only administrators can use to perform database server functions and maintenance functions, including those involving the execution of SQL and DDL statements.

Operating System
Action

UNIX or Windows

- Execute **oninit -j**.

User **informix** or a DBSA can use the **oninit -U** command to specify a list of administration mode users, as shown in this example:

```
oninit -U mark,ajay,carol
```

Users specified in the **oninit -U** list can connect for the period of time in which the server instance is active or until you run the **onmode -j -U** command to change the list of users who can connect to the server. Run the **onmode -j -U** command with a blank space instead of a name to remove all users in the list, as shown in this example:

```
oninit -U " "
```

Also see "Specifying Administration Mode Users with the ADMIN_MODE_USERS Configuration Parameter" on page 4-15.

Change From Quiescent to Online Mode

When you take the database server from quiescent mode to online mode, all sessions gain access.

If you have already taken the database server from online mode to quiescent mode and you are now returning the database server to online mode, any users who were interrupted in earlier processing must reselect their database and redeclare their cursors.

Operating System
Action

UNIX or Windows

- Execute **onmode -m**.

Windows only

- With the Services tool, choose the database server service and click **Continue**.

Change Gracefully from Online to Quiescent Mode

Take the database server gracefully from online mode to quiescent mode to restrict access to the database server without interrupting current processing.

After you perform this task, the database server sets a flag that prevents new sessions from gaining access to the database server. Current sessions are allowed to finish processing.

After you initiate the mode change, it cannot be cancelled. During the mode change from online to quiescent, the database server is considered to be in Shutdown mode.

Operating System
Action

UNIX or Windows

- Execute **onmode -s** or **onmode -sy.**

Windows only

- With the Services tool, choose the database server service and click **Pause**.

Change Immediately from Online to Quiescent Mode

Take the database server immediately from online mode to quiescent mode to restrict access to the database server as soon as possible. Work in progress can be lost.

A prompt asks for confirmation of the immediate shutdown. If you confirm, the database server sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates the session.

The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated.

The database server cleans up all sessions that the database server terminated. Active transactions are rolled back.

Operating System
Action

UNIX or Windows

- Execute **onmode -u** or **onmode -uy**.

  The **-y** option eliminates the need to confirm the prompt.

Change From Quiescent or Online to Administration Mode

When you move the database server from quiescent or online to administration mode, you move the server into a mode that only administrators can use.

If you begin in online mode, the database server automatically disconnects any users who are connected with any user ID that is not user **informix** and the users receive an error message. If a connection is terminated during a transaction, the database server rolls back the transaction.

Change to administration mode when you want to execute SQL and DLL commands when no other users are connected.

Operating System
Action

UNIX or Windows

- Execute **onmode -j**.

User **informix** or a DBSA can use the **onmode -j -U** option to grant individual users access to the database server in administration mode.

For example, run the following command to enable three individual users to connect to the database server and have database server access until the database server mode changes to offline, quiescent or online mode:

```
onmode -j -U mark,ajay,carol
```

Once connected, these individual users can execute any SQL or DDL commands. When the server is changed to administration mode, all sessions for users not identified in the **onmode -j -U** command lose their database server connection.

After initially running the **onmode -j -U** command, you can remove individuals by executing **onmode -j -U** and removing individual user names from the new list of names in the command, for example, by running:

```
onmode -j -U mark,carol
```

Run the **onmode -j -U** command with a blank space instead of a name to remove all users in the list, as shown in this example:

```
oninit -U " "
```

Also see "Specifying Administration Mode Users with the ADMIN_MODE_USERS Configuration Parameter" on page 4-15.

Change From Administration to Online Mode

When you move the database server from administration to online mode, all users can access the database server.

Operating System
Action

UNIX or Windows

- Execute **onmode -m**.

Change From Administration to Quiescent Mode

When you move the database server from administration to quiescent mode, you move the server into a mode that only administrators can use to perform maintenance functions that do not involve the execution of SQL and DDL statements.

Operating System
Action

UNIX or Windows

- Execute **onmode -s**.

Change From Any Mode Immediately to Offline Mode

You can take the database server immediately from any mode to offline mode.

A prompt asks for confirmation to go offline. If you confirm, the database server initiates a checkpoint request and sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates this session.

The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated.

After you take the database server to offline mode, restart the database server in quiescent, administration, or online mode. When you restart the database server, it performs a fast recovery to ensure that the data is logically consistent.

The database server cleans up all sessions that were terminated by the database server. Active transactions are rolled back.

Operating System
Action

UNIX or Windows

- Execute **onmode -k** or **onmode -ky**. The **-y** option eliminates the automatic prompt that confirms an immediate shutdown.

Windows only

- With the Services tool, choose the database server service and click **Stop**.

+ If the **onmode** command fails to shut down the database server, you can use the
+ **onclean** utility to force an immediate shutdown. For more information on the
+ **onclean** utility, see the *IBM Informix Dynamic Server Administrator's Reference*.

Specifying Administration Mode Users with the ADMIN_MODE_USERS Configuration Parameter

The ADMIN_MODE_USERS configuration parameter enables you to specify which users can connect to the database server in administration mode. Unlike the **oninit** and **onmode** commands that enable you to specify administration mode users until the server changes to offline, quiescent, or online mode, the ADMIN_MODE_USERS configuration parameter preserves a list of administration mode users indefinitely.

To create a list of administration mode users that is preserved in the ONCONFIG file, specify a comma-separated list of users as ADMIN_MODE_USERS configuration parameter values, for example, `mark,ajay,carol`.

To override ADMIN_MODE_USERS during a session, use the **onmode -wf** command, as shown in this example:

```
onmode -wf ADMIN_MODE_USERS=sharon,kalpana
```

The effect of the ADMIN_MODE_USERS configuration parameter is to add to the list of people permitted to access the server in administration mode. Those people listed in the **onmode** command line override those listed in the ONCONFIG file.

Part 2. Disk, Memory, and Process Management

Chapter 5. Virtual Processors and Threads

In This Chapter

This chapter describes virtual processors, explains how threads run within the virtual processors, and explains how the database server uses virtual processors and threads to improve performance.

Virtual Processors

Database server processes are called *virtual processors* because the way they function is similar to the way that a CPU functions in a computer. Just as a CPU runs multiple operating-system processes to service multiple users, a database server virtual processor runs multiple *threads* to service multiple SQL client applications.

A virtual processor is a process that the operating system schedules for processing.

Figure 5-1 illustrates the relationship of client applications to virtual processors. A small number of virtual processors serve a much larger number of client applications or queries.

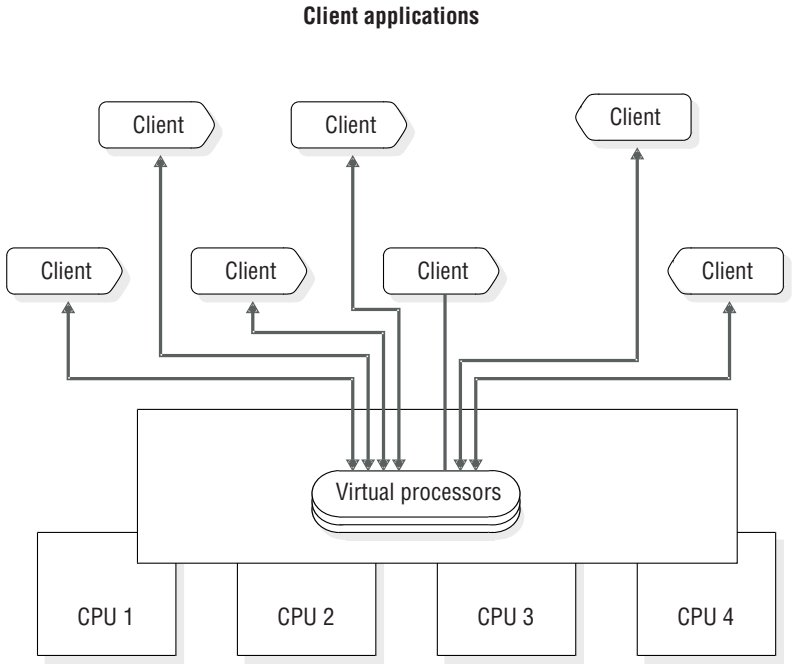


Figure 5-1. Virtual Processors

Threads

A thread is a task for a virtual processor in the same way that the virtual processor is a task for the CPU. The virtual processor is a task that the operating system schedules for execution on the CPU; a database server thread is a task that the

virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes, but they make fewer demands on the operating system.

Database server virtual processors are *multithreaded* because they run multiple concurrent threads.

The nature of threads is as follows.

Operating System	Description of Thread
UNIX	A thread is a task that the virtual processor schedules internally for processing.
Windows	A thread is a task that the virtual processor schedules internally for processing. Because the virtual processor is implemented as a Windows thread, database server threads run within Windows threads.

Important: Throughout this chapter, all references to "thread" refer to the threads created, scheduled, and destroyed by the database server. All references to "Windows threads" refer to the threads created, scheduled, and destroyed by Windows.

A virtual processor runs threads on behalf of SQL client applications (*session* threads) and also to satisfy internal requirements (*internal* threads). In most cases, for each connection by a client application, the database server runs one session thread. The database server runs internal threads to accomplish, among other things, database I/O, logging I/O, page cleaning, and administrative tasks. For cases in which the database server runs multiple session threads for a single client, refer to "Processing in Parallel" on page 5-5.

A *user thread* is a database server thread that services requests from client applications. User threads include session threads, called **sqlexec** threads, which are the primary threads that the database server runs to service client applications.

User threads also include a thread to service requests from the **onmode** utility, threads for recovery, B-tree scanner threads, and page-cleaner threads.

To display active user threads, use **onstat -u**. For more information on monitoring sessions and threads, refer to your *IBM Informix Dynamic Server Performance Guide*.

Types of Virtual Processors

Table 5-1 on page 5-3 shows the *classes* of virtual processors and the types of processing that they do. Each class of virtual processor is dedicated to processing certain types of threads.

The number of virtual processors of each class that you configure depends on the availability of physical processors (CPUs), hardware memory, and the database applications in use.

Table 5-1. Virtual-Processor Classes

Virtual-Processor Class	Category	Purpose
CPU	Central processing	Runs all session threads and some system threads. Runs thread for kernel asynchronous I/O (KAIO) where available. Can run a single poll thread, depending on configuration.
PIO	Disk I/O	Writes to the physical-log file (internal class) if it is in cooked disk space.
LIO	Disk I/O	Writes to the logical-log files (internal class) if they are in cooked disk space.
AIO	Disk I/O	Performs nonlogging disk I/O. If KAIO is used, AIO virtual processors perform I/O to cooked disk spaces.
SHM	Network	Performs shared memory communication.
TLI	Network	Uses the Transport Layer Interface (TLI) to perform network communication.
SOC	Network	Uses sockets to perform network communication.
OPT (UNIX)	Optical	Performs I/O to optical disk.
ADM	Administrative	Performs administrative functions.
ADT	Auditing	Performs auditing functions.
MSC	Miscellaneous	Services requests for system calls that require a very large stack.
CSM	Communications Support Module	Performs communications support service operations.
Encrypt	Encryption	Used by the database server when encryption or decryption functions are called.
classname	User defined	Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected. Specified with the VPCLASS configuration parameter. You must specify *classname*.
Java VP (JVP)	Java UDR	Executes Java UDRs. Contains the Java Virtual Machine (JVM).

Figure 5-2 on page 5-4 illustrates the major components and the extensibility of the database server.

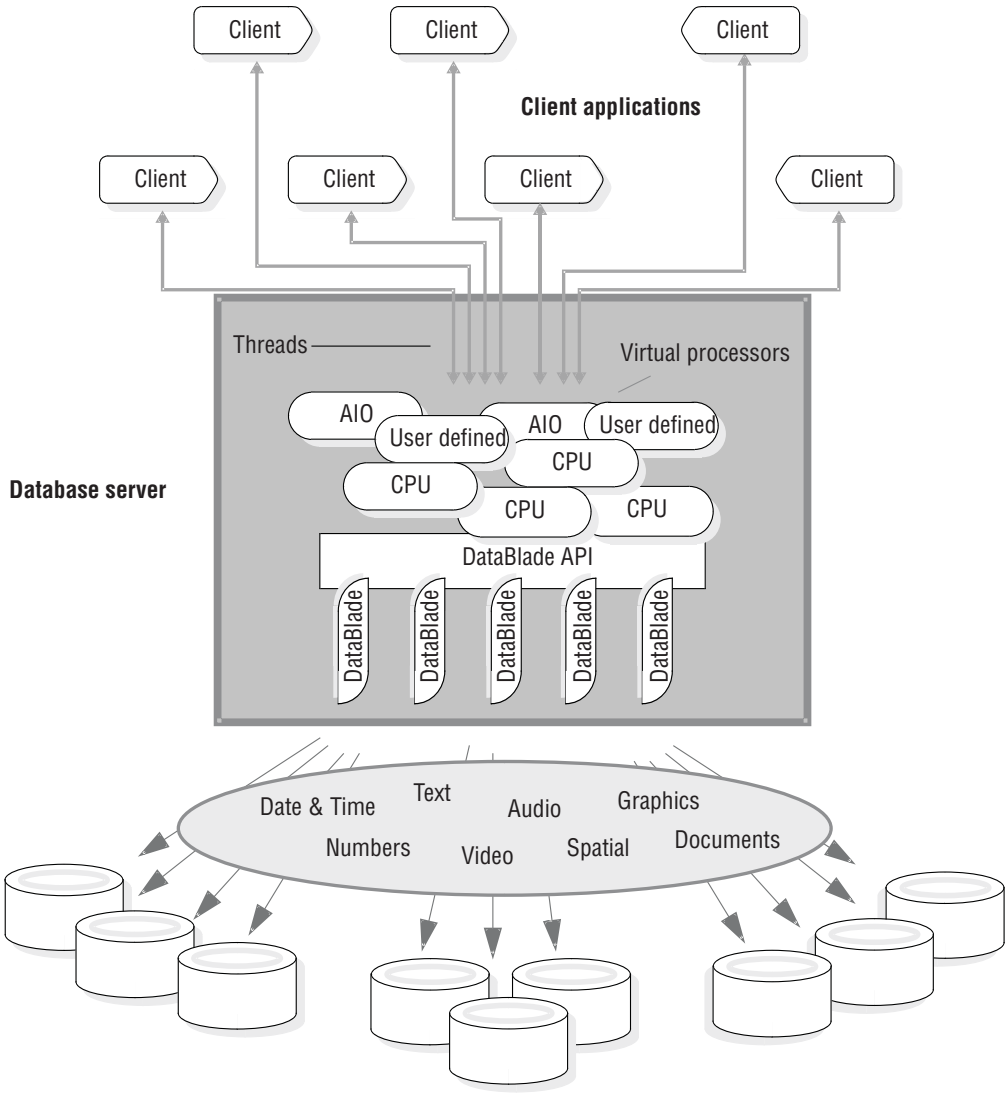


Figure 5-2. Database Server

Advantages of Virtual Processors

Compared to a database server process that services a single client application, the dynamic, multithreaded nature of a database server virtual processor provides the following advantages:

- Virtual processors can share processing.
- Virtual processors save memory and resources.
- Virtual processors can perform parallel processing.
- You can start additional virtual processors and terminate active CPU virtual processors while the database server is running.
- You can bind virtual processors to CPUs.

The following sections describe these advantages.

Sharing Processing

Virtual processors in the same class have identical code and share access to both data and processing queues in memory. Any virtual processor in a class can run any thread that belongs to that class.

Generally, the database server tries to keep a thread running on the same virtual processor because moving it to a different virtual processor can require some data from the memory of the processor to be transferred on the bus. When a thread is waiting to run, however, the database server can migrate the thread to another virtual processor because the benefit of balancing the processing load outweighs the amount of overhead incurred in transferring the data.

Shared processing within a class of virtual processors occurs automatically and is transparent to the database user.

Saving Memory and Resources

The database server is able to service a large number of clients with a small number of server processes compared to architectures that have one client process to one server process. It does so by running a thread, rather than a process, for each client.

Multithreading permits more efficient use of the operating-system resources because threads share the resources allocated to the virtual processor. All threads that a virtual processor runs have the same access to the virtual-processor memory, communication ports, and files. The virtual processor coordinates access to resources by the threads. Individual processes, on the other hand, each have a distinct set of resources, and when multiple processes require access to the same resources, the operating system must coordinate the access.

Generally, a virtual processor can switch from one thread to another faster than the operating system can switch from one process to another. When the operating system switches between processes, it must stop one process from running on the processor, save its current processing state (or context), and start another process. Both processes must enter and exit the operating-system kernel, and the contents of portions of physical memory might need to be replaced. Threads, on the other hand, share the same virtual memory and file descriptors. When a virtual processor switches from one thread to another, the switch is simply from one path of execution to another. The virtual processor, which is a process, continues to run on the CPU without interruption. For a description of how a virtual processor switches from one thread to another, refer to "Context Switching" on page 5-7.

Processing in Parallel

In the following cases, virtual processors of the CPU class can run multiple session threads, working in parallel, for a single client:

- Index building
- Sorting
- Recovery
- Scanning
- Joining
- Aggregation
- Grouping
- User-defined-routine (UDR) execution

For more information on parallel UDR execution, refer to *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Figure 5-3 illustrates parallel processing. When a client initiates index building, sorting, or logical recovery, the database server spawns multiple threads to work on the task in parallel, using as much of the computer resources as possible. While one thread is waiting for I/O, another can be working.

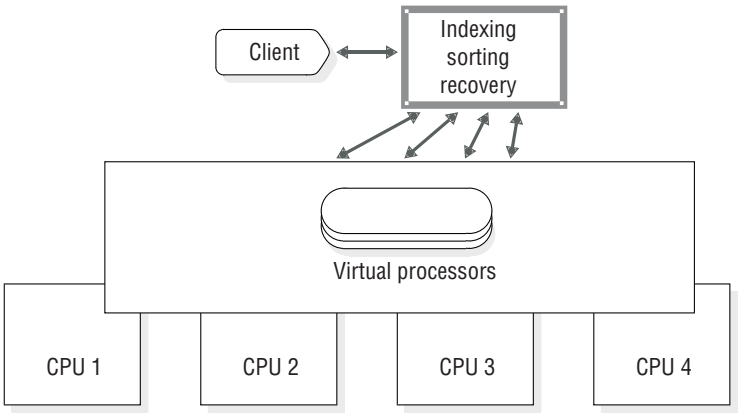


Figure 5-3. Parallel Processing

Adding and Dropping Virtual Processors in Online Mode

You can add virtual processors to meet increasing demands for service while the database server is running. For example, if the virtual processors of a class become compute bound or I/O bound (meaning that CPU work or I/O requests are accumulating faster than the current number of virtual processors can process them), you can start additional virtual processors for that class to distribute the processing load further.

You can add virtual processors for any of the classes while the database server is running. For more information, see "Adding Virtual Processors in Online Mode" on page 6-3.

Windows Only:

In Windows, you cannot drop a virtual processor of any class.

While the database server is running, you can drop virtual processors of the CPU or a user-defined class. For more information, see "Setting Virtual-Processor Configuration Parameters" on page 6-1.

Binding Virtual Processors to CPUs

Some multiprocessor systems allow you to bind a process to a particular CPU. This feature is called *processor affinity*.

On multiprocessor computers for which the database server supports processor affinity, you can bind CPU virtual processors to specific CPUs in the computer. When you bind a CPU virtual processor to a CPU, the virtual processor runs exclusively on that CPU. This operation improves the performance of the virtual processor because it reduces the amount of switching between processes that the operating system must do. Binding CPU virtual processors to specific CPUs also

enables you to isolate database work on specific processors on the computer, leaving the remaining processors free for other work. Only CPU virtual processors can be bound to CPUs.

For information on how to assign CPU virtual processors to hardware processors, refer to "Using Processor Affinity" on page 5-13.

How Virtual Processors Service Threads

At a given time, a virtual processor can run only one thread. A virtual processor services multiple threads concurrently by switching between them. A virtual processor runs a thread until it yields. When a thread yields, the virtual processor switches to the next thread that is ready to run. The virtual processor continues this process, eventually returning to the original thread when that thread is ready to continue. Some threads complete their work, and the virtual processor starts new threads to process new work. Because a virtual processor continually switches between threads, it can keep the CPU processing continually. The speed at which processing occurs produces the appearance that the virtual processor processes multiple tasks simultaneously and, in effect, it does.

Running multiple concurrent threads requires scheduling and synchronization to prevent one thread from interfering with the work of another. Virtual processors use the following structures and methods to coordinate concurrent processing by multiple threads:

- Control structures
- Context switching
- Stacks
- Queues
- Mutexes

This section describes how virtual processors use these structures and methods.

Control Structures

When a client connects to the database server, the database server creates a *session* structure, which is called a *session control block*, to hold information about the connection and the user. A session begins when a client connects to the database server, and it ends when the connection terminates.

Next, the database server creates a thread structure, which is called a *thread-control block* (TCB) for the session, and initiates a primary thread (**sqlexec**) to process the client request. When a thread *yields*—that is, when it pauses and allows another thread to run—the virtual processor saves information about the state of the thread in the thread-control block. This information includes the content of the process system registers, the program counter (address of the next instruction to execute), and the stack pointer. This information constitutes the *context* of the thread.

In most cases, the database server runs one primary thread per session. In cases where it performs parallel processing, however, it creates multiple session threads for a single client, and, likewise, multiple corresponding thread-control blocks.

Context Switching

A virtual processor switches from running one thread to running another one by *context switching*. The database server does not preempt a running thread, as the

operating system does to a process, when a fixed amount of time (time-slice) expires. Instead, a thread yields at one of the following points:

- A predetermined point in the code
- When the thread can no longer execute until some condition is met

When the amount of processing required to complete a task would cause other threads to wait for an undue length of time, a thread yields at a predetermined point. The code for such long-running tasks includes calls to the yield function at strategic points in the processing. When a thread performs one of these tasks, it yields when it encounters a yield function call. Other threads in the ready queue then get a chance to run. When the original thread next gets a turn, it resumes executing code at the point immediately after the call to the yield function. Predetermined calls to the yield function allow the database server to interrupt threads at points that are most advantageous for performance.

A thread also yields when it can no longer continue its task until some condition occurs. For example, a thread yields when it is waiting for disk I/O to complete, when it is waiting for data from the client, or when it is waiting for a lock or other resource.

When a thread yields, the virtual processor saves its context in the thread-control block. Then the virtual processor selects a new thread to run from a queue of ready threads, loads the context of the new thread from its thread-control block, and begins executing at the new address in the program counter. Figure 5-4 illustrates how a virtual processor accomplishes a context switch.

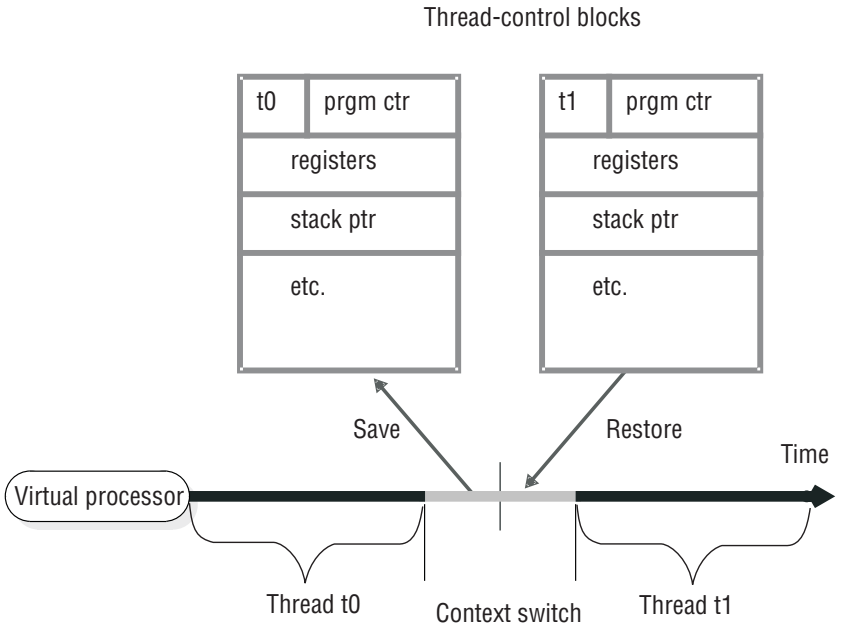


Figure 5-4. Context Switch: How a Virtual Processor Switches from One Thread to Another

Stacks

The database server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area is called the *stack*. For information on how to set the size of the stack, refer to "Stacks" on page 7-17.

The stack enables a virtual processor to protect the nonshared data of a thread from being overwritten by other threads that concurrently execute the same code. For example, if several client applications concurrently perform SELECT statements, the session threads for each client execute many of the same functions in the code. If a thread did not have a private stack, one thread could overwrite local data that belongs to another thread within a function.

When a virtual processor switches to a new thread, it loads a stack pointer for that thread from a field in the thread-control block. The stack pointer stores the beginning address of the stack. The virtual processor can then specify offsets to the beginning address to access data within the stack. Figure 5-5 illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.

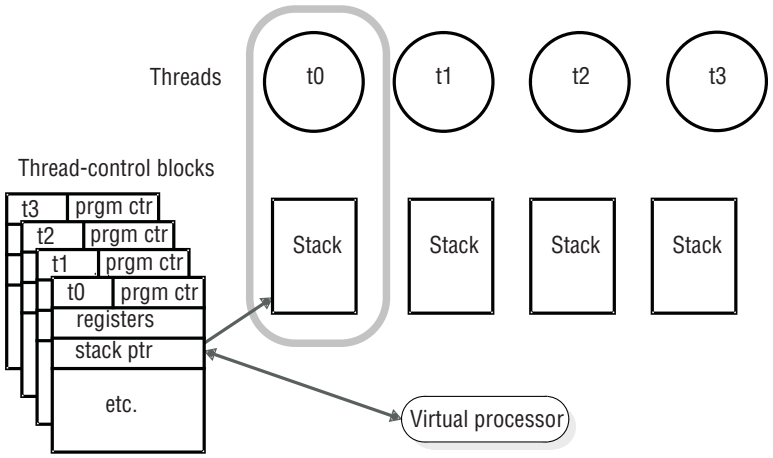


Figure 5-5. Virtual Processors Segregate Nonshared Data for Each User

Queues

The database server uses three types of queues to schedule the processing of multiple, concurrently running threads:

- Ready queues
- Sleep queues
- Wait queues

Virtual processors of the same class share queues. This fact, in part, enables a thread to migrate from one virtual processor in a class to another when necessary.

Ready Queues

Ready queues hold threads that are ready to run when the current (running) thread yields. When a thread yields, the virtual processor picks the next thread with the appropriate priority from the ready queue. Within the queue, the virtual processor processes threads that have the same priority on a first-in-first-out (FIFO) basis.

On a multiprocessor computer, if you notice that threads are accumulating in the ready queue for a class of virtual processors (indicating that work is accumulating faster than the virtual processor can process it), you can start additional virtual processors of that class to distribute the processing load. For information on how to monitor the ready queues, refer to "Monitoring Virtual Processors" on page 6-5. For information on how to add virtual processors while the database server is in online mode, refer to "Adding Virtual Processors in Online Mode" on page 6-3.

Sleep Queues

Sleep queues hold the contexts of threads that have no work to do at a particular time. A thread is put to sleep either for a specified period of time or *forever*.

The administration class (ADM) of virtual processors runs the system timer and special utility threads. Virtual processors in this class are created and run automatically. No configuration parameters impact this class of virtual processors.

The ADM virtual processor wakes up threads that have slept for the specified time. A thread that runs in the ADM virtual processor checks on sleeping threads at one-second intervals. If a sleeping thread has slept for its specified time, the ADM virtual processor moves it into the appropriate ready queue. A thread that is sleeping for a specified time can also be explicitly awakened by another thread.

A thread that is sleeping *forever* is awakened when it has more work to do. For example, when a thread that is running on a CPU virtual processor needs to access a disk, it issues an I/O request, places itself in a sleep queue for the CPU virtual processor, and yields. When the I/O thread notifies the CPU virtual processor that the I/O is complete, the CPU virtual processor schedules the original thread to continue processing by moving it from the sleep queue to a ready queue. Figure 5-6 illustrates how the database server threads are queued to perform database I/O.

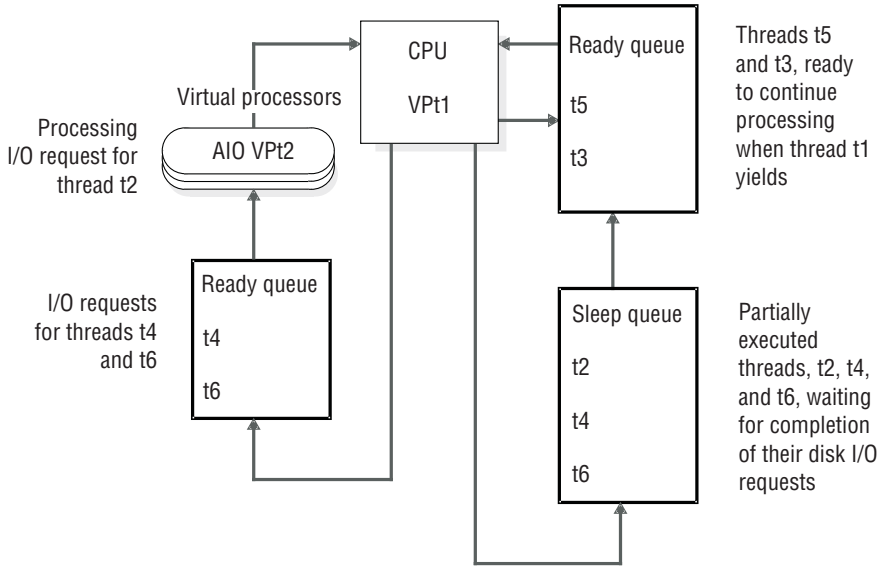


Figure 5-6. How Database Server Threads Are Queued to Perform Database I/O

Wait Queues

Wait queues hold threads that need to wait for a particular event before they can continue to run. For example, wait queues coordinate access to shared data by threads. When a user thread tries to acquire the logical-log latch but finds that the latch is held by another user, the thread that was denied access puts itself in the logical-log wait queue. When the thread that owns the lock is ready to release the latch, it checks for waiting threads, and, if threads are waiting, it wakes up the next thread in the wait queue.

Mutexes

A mutex (*mut*ually *ex*clusive), also referred to as a *latch*, is a latching mechanism that the database server uses to synchronize access by multiple threads to shared resources. Mutexes are similar to semaphores, which some operating systems use to regulate access to shared data by multiple *processes*. However, mutexes permit a greater degree of parallelism than semaphores.

A mutex is a variable that is associated with a shared resource such as a buffer. A thread must acquire the mutex for a resource before it can access the resource. Other threads are excluded from accessing the resource until the owner releases it. A thread acquires a mutex, after a mutex becomes available, by setting it to an in-use state. The synchronization that mutexes provide ensures that only one thread at a time writes to an area of shared memory.

For information on monitoring mutexes, refer to "Monitoring the Shared-Memory Profile and Latches" on page 8-7.

Virtual-Processor Classes

A virtual processor of a given class can run only threads of that class. This section describes the types of threads, or the types of processing, that each class of virtual processor performs. It also explains how to determine the number of virtual processors that you need to run for each class.

CPU Virtual Processors

The CPU virtual processor runs all session threads (the threads that process requests from SQL client applications) and some internal threads. Internal threads perform services that are internal to the database server. For example, a thread that listens for connection requests from client applications is an internal thread.

Each CPU virtual processor can have a private memory cache associated with it. Each private memory cache block consists of 1 to 32 memory pages, where each memory page is 4096 bytes. The database server uses the private memory cache to improve access time to memory blocks. Use the VP_MEMORY_CACHE_KB configuration parameter to enable a private memory cache and specify information about the memory cache. For more information, see the *IBM Informix Dynamic Server Administrator's Reference* and the *IBM Informix Dynamic Server Performance Guide*.

Determining the Number of CPU Virtual Processors Needed

The right number of CPU virtual processors is the number at which they are all kept busy but not so busy that they cannot keep pace with incoming requests. You should not allocate more CPU virtual processors than the number of hardware processors in the computer.

To evaluate the performance of the CPU virtual processors while the database server is running, repeat the following command at regular intervals over a set period of time:

```
onstat -g glo
```

If the accumulated *usercpu* and *syscpu* times, taken together, approach 100 percent of the actual elapsed time for the period of the test, add another CPU virtual processor if you have a CPU available to run it.

For information on deciding how many CPU virtual processors you need, see "Running Poll Threads on CPU or Network Virtual Processors" on page 5-22.

The VPCLASS parameter allows you to specify all of the following information:

- The number of virtual processors to start initially for a class
- The maximum number of virtual processors to run for the class
- Processor affinity for CPU class virtual processors
- Disabling of priority aging, if applicable

Note: You should use the VPCLASS parameter instead of the AFF_SPROC, AFFNPROCS, NOAGE, NUMCPUVPS, and NUMAIOVPS parameters. The VPCLASS parameter cannot be used in combination with these parameters. If the ONCONFIG file contains a NUMCPUVPS parameter, for example, you receive an error message if the file also contains a VPCLASS *cpu* parameter. For information about the VPCLASS configuration parameter, refer to the *IBM Informix Administrator's Reference*.

In addition to considering the number of CPUs in the computer and the number of users who will connect to the database server, also consider the fact that user-defined routines and DataBlade modules, which are collections of user-defined routines, run on either CPU virtual processors or user-defined virtual processors. For information on how you assign a user-defined routine to a virtual processor, refer to "Assigning a UDR to a User-Defined Virtual-Processor Class" on page 5-16.

Running on a Multiprocessor Computer

If you are running multiple CPU virtual processors on a multiprocessor computer, set the MULTIPROCESSOR parameter in the ONCONFIG file to 1. When you set MULTIPROCESSOR to 1, the database server performs locking in a manner that is appropriate for a multiprocessor computer. For information on setting multiprocessor mode, refer to the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

Running on a Single-Processor Computer

If you are running the database server on a single-processor computer, set the MULTIPROCESSOR configuration parameter to `0`. To run the database server with only one CPU virtual processor, set the SINGLE_CPU_VP parameter to 1.

Setting MULTIPROCESSOR to `0` enables the database server to bypass the locking that is required for multiple processes on a multiprocessor computer. For information on the MULTIPROCESSOR configuration parameter, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

Setting SINGLE_CPU_VP to 1 allows the database server to bypass some of the mutex calls that it normally makes when it runs multiple CPU virtual processors. For information on setting the SINGLE_CPU_VP parameter, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

Important: Setting *VPCLASS num* to *1* and SINGLE_CPU_VP to *0* does not reduce the number of mutex calls, even though the database server starts only one CPU virtual processor. You must set SINGLE_CPU_VP to *1* to reduce the amount of latching that is performed when you run a single CPU virtual processor.

Setting the SINGLE_CPU_VP parameter to 1 imposes two important restrictions on the database server, as follows:

- Only one CPU virtual processor is allowed.

  You cannot add CPU virtual processors while the database server is in online mode.
- No user-defined classes are allowed. (However, users can still define routines that run directly on the CPU VP.)

For more information, see "Adding Virtual Processors in Online Mode" on page 6-3.

Adding and Dropping CPU Virtual Processors in Online Mode

You can add or drop CPU class virtual processors while the database server is online. For instructions on how to do this, see "Adding Virtual Processors in Online Mode" on page 6-3 and "Dropping CPU and User-Defined Virtual Processors" on page 6-4.

Preventing Priority Aging

Some operating systems lower the priority of long-running processes as they accumulate processing time. This feature of the operating system is called *priority aging*. Priority aging can cause the performance of database server processes to decline over time. In some cases, however, the operating system allows you to disable this feature and keep long-running processes running at a high priority.

To determine if priority aging is available on your computer, check the machine notes file, that comes with your installation and is described in the Introduction to this guide.

If your operating system allows you to disable priority aging, You can disable it by specifying `noage` for the priority entry in the VPCLASS configuration parameter. For more information, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

Using Processor Affinity

The database server supports automatic binding of CPU virtual processors to processors on multiprocessor computers that support *processor affinity*. Your database server distribution includes a machine notes file that contains information on whether your database server version supports this feature. When you assign a CPU virtual processor to a specific CPU, the virtual processor runs only on that CPU, but other processes also can run on that CPU.

Use the VPCLASS configuration parameter with the *aff* option to implement processor affinity on multiprocessor computers that support it.

Figure 5-7 on page 5-14 illustrates the concept of processor affinity.

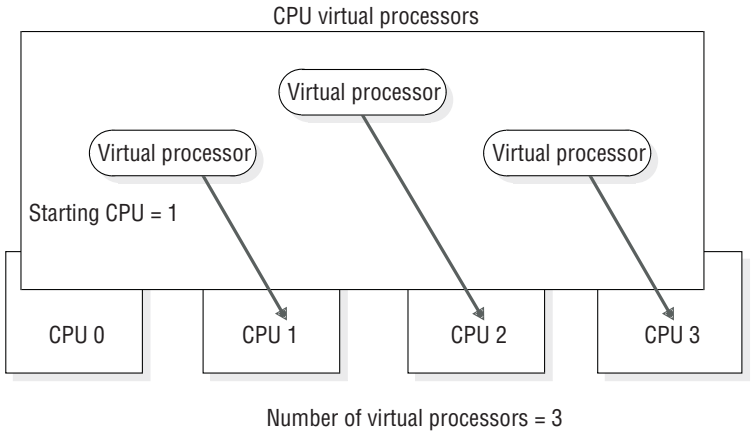


Figure 5-7. Processor Affinity

UNIX Only:

To see if processor affinity is supported on your UNIX platform, refer to the machine notes file.

Setting Processor Affinity with the VPCLASS Configuration Parameter:

To set processor affinity with the VPCLASS configuration parameter, you can specify individual processors or ranges of processors that you want to assign the virtual processors. When specifying a range of processors, you can also specify an incremental value with the range that indicates which CPUs in the range should be assigned to the virtual processors. For example, you can specify that the virtual processors are assigned to every other CPU in the range 0-6, starting with CPU 0.

```
VPCLASS CPU,num=4,aff=(0-6/2)
```

The virtual processors are assigned to CPUs 0, 2, 4, 6.

If you specify `VPCLASS CPU,num=4,aff=(1-10/3)`, the virtual processors are assigned to every third CPU in the range 1-10, starting with CPU 1. The virtual processors are assigned to CPUs 1, 4, 7, 10.

When you specify more than one value or range, the values and ranges do not have to be incremental or in any particular order. For example you can specify `aff=(8,12,7-9,0-6/2)`.

The database server assigns CPU virtual processors to CPUs in a circular pattern, starting with the first processor number that you specify in the *aff* option. If you specify a larger number of CPU virtual processors than physical CPUs, the database server continues to assign CPU virtual processors starting with the first CPU. For example, suppose you specify the following VPCLASS settings:

```
VPCLASS cpu,num=8,aff=(4-7)
```

The database server makes the following assignments:

- CPU virtual processor number `0` to CPU `4`
- CPU virtual processor number `1` to CPU `5`
- CPU virtual processor number `2` to CPU `6`
- CPU virtual processor number `3` to CPU `7`
- CPU virtual processor number `4` to CPU `4`

- CPU virtual processor number 5 to CPU 5
- CPU virtual processor number 6 to CPU 6
- CPU virtual processor number 7 to CPU 7

For more information, see the VPCLASS configuration parameter in the *IBM Informix Dynamic Server Administrator's Reference*.

User-Defined Classes of Virtual Processors

You can define special classes of virtual processors to run user-defined routines or to run a DataBlade module. User-defined routines are typically written to support user-defined data types. If you do not want a user-defined routine to run in the CPU class, which is the default, you can assign it to a user-defined class of virtual processors (VPs). User-defined classes of virtual processors are also called *extension virtual processors*.

This section provides the following information about user-defined virtual processors:

- When to execute a C-language UDR in a user-defined VP instead of in the CPU VP
- How to assign a C-language UDR to a particular user-defined VP class
- How to add and drop user-defined VPs when the database server is in online mode

Determining the Number of User-Defined Virtual Processors Needed

You can specify as many user-defined virtual processors as your operating system will allow. If you run many UDRs or parallel PDQ queries with UDRs, you should configure more user-defined virtual processors.

Using User-Defined Virtual Processors

User-defined classes of virtual processors protects the database server from *ill-behaved* user-defined routines. An ill-behaved user-defined routine has at least one of the following characteristics:

- Does not yield control to other threads
- Makes blocking operating-system calls
- Modifies the global VP state

A well-behaved C-language UDR has none of these characteristics. Execute only well-behaved C-language UDRs in a CPU VP.

Warning: Execution of an ill-behaved routine in a CPU VP can cause serious interference with the operation of the database server, possibly causing it to fail or behave erratically. In addition, the routine itself might not produce correct results.

To ensure safe execution, assign any ill-behaved user-defined routines to a user-defined class of virtual processors. User-defined VPs remove the following programming restrictions on the CPU VP class:

- The need to yield the processor regularly
- The need to eliminate blocking I/O calls

Functions that run in a user-defined virtual-processor class need not yield the processor, and they might issue direct file-system calls that block further processing by the virtual processor until the I/O is complete.

The normal processing of user queries is not affected by ill-behaved traits of a C-language UDR because these UDRs do not execute in CPU virtual processors. For a more detailed discussion of ill-behaved routines, refer to the *IBM Informix DataBlade API Programmer's Guide*.

Specifying a User-Defined Virtual Processor

The VPCLASS parameter with the *vpclass* option defines a user-defined VP class. You also can specify a nonyielding user-defined virtual processor. For more information, see "Specifying VPCLASS" on page 6-2 and the configuration parameters chapter of the *IBM Informix Dynamic Server Administrator's Reference*.

Assigning a UDR to a User-Defined Virtual-Processor Class

The SQL CREATE FUNCTION statement registers a user-defined routine. For example, the following CREATE FUNCTION statement registers the user-defined routine, **GreaterThanEqual()**, and specifies that calls to this routine should be executed by the user-defined VP class named UDR:

```
CREATE FUNCTION GreaterThanEqual(ScottishName, ScottishName)
   RETURNS boolean
   WITH (CLASS = UDR  )
   EXTERNAL NAME '/usr/lib/objects/udrs.so'
   LANGUAGE C
```

To execute this function, the ONCONFIG file must include a VPCLASS parameter that defines the UDR class. If not, calls to the **GreaterThanEqual** function will fail.

Tip: The CLASS routine modifier can specify any name for the VP class. This class name need not exist when you register the UDR. However, when you try to run a UDR that specifies a user-defined VP class for its execution, this class must exist and have virtual processors assigned to it.

To configure the UDR class, include a line similar to the following one in the ONCONFIG file. This line configures the UDR class with two virtual processors and with no priority aging.

```
VPCLASS     UDR  ,num=2,noage
```

The preceding line defines the UDR VP class as a yielding VP class; that is, this VP class allows the C-language UDR to yield to other threads that need access to the UDR VP class. For more information on how to use the VPCLASS configuration parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

For more information on the CREATE FUNCTION statement, refer to the *IBM Informix Guide to SQL: Syntax*.

Adding and Dropping User-Defined Virtual Processors in Online Mode

You can add or drop virtual processors in a user-defined class while the database server is online. For instructions on how to do this, refer to "Adding Virtual Processors in Online Mode" on page 6-3 and "Dropping CPU and User-Defined Virtual Processors" on page 6-4.

Java Virtual Processors

Java UDRs and Java applications execute on specialized virtual processors, called *Java virtual processors* (JVPs). A JVP embeds a Java virtual machine (JVM) in its code. A JVP has the same capabilities as a CPU VP in that it can process complete SQL queries.

You can specify as many JVPs as your operating system will allow. If you run many Java UDRs or parallel PDQ queries with Java UDRs, you should configure more JVPs. For more information about UDRs written in Java, refer to *J/Foundation Developer's Guide*.

Use the VPCLASS configuration parameter with the `jvp` keyword to configure JVPs. For more information, see the configuration parameters chapter in the *IBM Informix Dynamic Server Administrator's Reference*.

Disk I/O Virtual Processors

The following classes of virtual processors perform disk I/O:

- PIO (physical-log I/O)
- LIO (logical-log I/O)
- AIO (asynchronous I/O)
- CPU (kernel-asynchronous I/O)

The PIO class performs all I/O to the physical-log file, and the LIO class performs all I/O to the logical-log files, *unless* those files reside in raw disk space and the database server has implemented KAIO.

On operating systems that do not support KAIO, the database server uses the AIO class of virtual processors to perform database I/O that is not related to physical or logical logging.

The database server uses the CPU class to perform KAIO when it is available on a platform. If the database server implements KAIO, a KAIO thread performs all I/O to raw disk space, including I/O to the physical and logical logs.

UNIX Only:

To find out if your UNIX platform supports KAIO, refer to the machine notes file.

Windows Only:

Windows supports KAIO.

For more information about nonlogging I/O, refer to "Asynchronous I/O" on page 5-18.

I/O Priorities

In general, the database server prioritizes disk I/O by assigning different types of I/O to different classes of virtual processors and by assigning priorities to the nonlogging I/O queues. Prioritizing ensures that a high-priority log I/O, for example, is never queued behind a write to a temporary file, which has a low priority. The database server prioritizes the different types of disk I/O that it performs, as Table 5-2 on page 5-18 shows.

Table 5-2. How Database Server Prioritizes Disk I/O

Priority	Type of I/O	VP Class
1st	Logical-log I/O	CPU or LIO
2nd	Physical-log I/O	CPU or PIO
3rd	Database I/O	CPU or AIO
3rd	Page-cleaning I/O	CPU or AIO
3rd	Read-ahead I/O	CPU or AIO

Logical-Log I/O

The LIO class of virtual processors performs I/O to the logical-log files in the following cases:

- KAIO is not implemented.
- The logical-log files are in cooked disk space.

Only when KAIO is implemented and the logical-log files are in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the logical log.

The logical-log files store the data that enables the database server to roll back transactions and recover from system failures. I/O to the logical-log files is the highest priority disk I/O that the database server performs.

If the logical-log files are in a dbspace that *is not* mirrored, the database server runs only one LIO virtual processor. If the logical-log files are in a dbspace that *is* mirrored, the database server runs two LIO virtual processors. This class of virtual processors has no parameters associated with it.

Physical-Log I/O

The PIO class of virtual processors performs I/O to the physical-log file in the following cases:

- KAIO is not implemented.
- The physical-log file is stored in buffered-file chunks.

Only when KAIO is implemented and the physical-log file is in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the physical log. The physical-log file stores *before-image*s of dbspace pages that have changed since the last *checkpoint*. (For more information on checkpoints, refer to "Checkpoints" on page 15-4.) At the start of recovery, prior to processing transactions from the logical log, the database server uses the physical-log file to restore before-images to dbspace pages that have changed since the last checkpoint. I/O to the physical-log file is the second-highest priority I/O after I/O to the logical-log files.

If the physical-log file is in a dbspace that *is not* mirrored, the database server runs only one PIO virtual processor. If the physical-log file is in a dbspace that *is* mirrored, the database server runs two PIO virtual processors. This class of virtual processors has no parameters associated with it.

Asynchronous I/O

The database server performs database I/O asynchronously, meaning that I/O is queued and performed independently of the thread that requests the I/O.

Performing I/O asynchronously allows the thread that makes the request to continue working while the I/O is being performed.

The database server performs all database I/O asynchronously, using one of the following facilities:

- AIO virtual processors
- KAIO on platforms that support it

Database I/O includes I/O for SQL statements, read-ahead, page cleaning, and checkpoints, as well as other I/O.

Kernel-Asynchronous I/O: The database server uses KAIO when the following conditions exist:

- The computer and operating system support it.
- A performance gain is realized.
- The I/O is to raw disk space.

The database server implements KAIO by running a KAIO thread on the CPU virtual processor. The KAIO thread performs I/O by making system calls to the operating system, which performs the I/O independently of the virtual processor. The KAIO thread can produce better performance for disk I/O than the AIO virtual processor can, because it does not require a switch between the CPU and AIO virtual processors.

UNIX Only:

Dynamic Server implements KAIO when Dynamic Server ports to a platform that supports this feature. The database server administrator does not configure KAIO. To see if KAIO is supported on your platform, see the machine notes file.

Linux Only:

Kernel asynchronous I/O (KAIO) is enabled by default. You can disable this by specifying that `KAIOOFF=1` in the environment of the process that brings up the server.

On Linux, there is a system-wide limit of the maximum number of parallel KAIO requests. The **/proc/sys/fs/aio-max-nr** file contains this value. The Linux system administrator can increase the value, for example, by using this command:

```
# echo new_value > /proc/sys/fs/aio-max-nr
```

The current number of allocated requests of all operating system processes is visible in the **/proc/sys/fs/aio-nr** file.

By default, Dynamic Version allocates half of the maximum number of requests and assigns them equally to the number of configured CPU virtual processors. You can use the environment variable KAIOON to control the number of requests allocated per CPU virtual processor. Do this by setting KAIOON to the required value before starting Dynamic Server.

The minimum value for KAIOON is `100`. If Linux is about to run out of KAIO resources, for example when dynamically adding many CPU virtual processors, warnings are printed in the **online.log** file. If this happens, the Linux system administrator should add KAIO resources as described above.

AIO Virtual Processors: If the platform does not support KAIO or if the I/O is to buffered-file chunks, the database server performs database I/O through the AIO class of virtual processors. All AIO virtual processors service all I/O requests equally within their class.

The database server assigns each disk chunk a queue, sometimes known as a *gfd queue*, based on the filename of the chunk. The database server orders I/O requests within a queue according to an algorithm that minimizes disk-head movement. The AIO virtual processors service queues that have work pending in round-robin fashion.

All other non-chunk I/O is queued in the AIO queue.

Use the VPCLASS parameter with the *aio* keyword to specify the number of AIO virtual processors that the database server starts initially. For information about VPCLASS, refer to the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

You can start additional AIO virtual processors while the database server is in online mode. For more information, refer to "Adding Virtual Processors in Online Mode" on page 6-3.

You cannot drop AIO virtual processors while the database server is in online mode.

Automatic Increasing and Decreasing of AIO Virtual Processors:

The AUTO_AIOVPS configuration parameter enables or disables the database server to automatically increase the number of AIO VPS and flusher threads when the server detects that AIO VPs are not keeping up with the I/O workload. Disable this parameter if you want to manually adjust the number. For details on setting this parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

Number of AIO Virtual Processors Needed:

The goal in allocating AIO virtual processors is to allocate enough of them so that the lengths of the I/O request queues are kept short; that is, the queues have as few I/O requests in them as possible. When the *gfd* queues are consistently short, it indicates that I/O to the disk devices is being processed as fast as the requests occur.

The **onstat -g ioq** command displays the length and other statistics about I/O queues. This command allows you to monitor the length of the *gfd* queues for the AIO virtual processors. For more information, refer to "Monitoring Virtual Processors" on page 6-5 and information on monitoring virtual processors in the *IBM Informix Dynamic Server Performance Guide*.

One AIO virtual processor might be sufficient:

- If the database server implements kernel asynchronous I/O (KAIO) on your platform and all of your dbspaces are composed of raw disk space
- If your file system supports direct I/O for the page size used for the dbspace chunk and you use direct I/O

Allocate two AIO virtual processors per active dbspace that is composed of buffered file chunks.

- If the database server implements KAIO, but you are using some buffered files for chunks
- IF KAIO is not supports by the system for chunks.

If KAIO is *not* implemented on your platform, allocate two AIO virtual processors for each disk that the database server accesses frequently.

If you use cooked files and if you enable direct I/O using the DIRECT_IO configuration parameter, you might be able to reduce the number of AIO virtual processors.

If the database server implements KAIO and you enabled direct I/O using the DIRECT_IO configuration parameter, Dynamic Server will attempt to use KAIO, so you probably do not need more than one AIO virtual processor. However, even when direct I/O is enabled, if the file system does not support either direct I/O or KAIO, you still need to allocate two AIO virtual processors for every active dbspace that is composed of buffered file chunks or does not use KAIO.

Temporary dbspaces do not use direct I/O. If you have temporary dbspaces, you will probably need more than one AIO virtual processors.

Allocate enough AIO virtual processors to accommodate the peak number of I/O requests. Generally, it is not detrimental to allocate too many AIO virtual processors.

You can use the AUTO_AIOVPS configuration parameter to enable the database server to automatically increase the number of AIO virtual processors and page-cleaner threads when the server detects that AIO virtual processors are not keeping up with the I/O workload.

Network Virtual Processors

As explained in Chapter 3, "Client/Server Communications," on page 3-1, a client can connect to the database server in the following ways:

- Through a network connection
- Through a pipe
- Through shared memory

The network connection can be made by a client on a remote computer or by a client on the local computer mimicking a connection from a remote computer (called a *local-loopback connection*).

Specifying Network Connections

In general, the DBSERVERNAME and DBSERVERALIASES parameters define dbservernames that have corresponding entries in the **sqlhosts** file or registry. Each dbservername parameter in **sqlhosts** has a **nettype** entry that specifies an interface/protocol combination. The database server runs one or more *poll threads* for each unique **nettype** entry. For a description of the **nettype** field, refer to "The Connection Type Field" on page 3-18.

The NETTYPE configuration parameter provides optional configuration information for an interface/protocol combination. It allows you to allocate more than one poll thread for an interface/protocol combination and also designate the virtual-processor class (CPU or NET) on which the poll threads run.

For a complete description of the NETTYPE configuration parameter, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

Running Poll Threads on CPU or Network Virtual Processors

Poll threads can run either on CPU virtual processors or on network virtual processors. In general, and particularly on a single-processor computer, poll threads run more efficiently on CPU virtual processors. This might not be true, however, on a multiprocessor computer with a large number of remote clients.

The NETTYPE parameter has an optional entry, called `vp class`, that allows you to specify either CPU or NET, for CPU or network virtual-processor classes, respectively.

If you do not specify a virtual processor class for the interface/protocol combination (poll threads) associated with the DBSERVERNAME variable, the class defaults to CPU. The database server assumes that the interface/protocol combination associated with DBSERVERNAME is the primary interface/protocol combination and that it should be the most efficient.

For other interface/protocol combinations, if no `vp class` is specified, the default is NET.

While the database server is in online mode, you cannot drop a CPU virtual processor that is running a poll thread.

Note: You should carefully distinguish between poll threads for network connections and poll threads for shared memory connections, which should run one per CPU virtual processor. TCP connections should only be in network virtual processors, and you should only have the minimum needed to maintain responsiveness. Shared memory connections should only be in CPU virtual processors and should run in every CPU virtual processor.

Specifying the Number of Networking Virtual Processors

Each poll thread requires a separate virtual processor, so you indirectly specify the number of networking virtual processors when you specify the number of poll threads for an interface/protocol combination and specify that they are to be run by the NET class. If you specify CPU for the `vp class`, you must allocate a sufficient number of CPU virtual processors to run the poll threads. If the database server does not have a CPU virtual processor to run a CPU poll thread, it starts a network virtual processor of the specified class to run it.

For most systems, one poll thread and consequently one virtual processor per network interface/protocol combination is sufficient. For systems with 200 or more network users, running additional network virtual processors might improve throughput. In this case, you need to experiment to determine the optimal number of virtual processors for each interface/protocol combination.

Specifying Listen and Poll Threads for the Client/Server Connection

When you start the database server, the **oninit** process starts an internal thread, called a *listen thread*, for each dbservername that you specify with the DBSERVERNAME and DBSERVERALIASES parameters in the ONCONFIG file. To specify a listen port for each of these dbservername entries, assign it a unique combination of **hostname** and **service name** entries in **sqlhosts**. For example, the

sqlhosts file or registry entry shown in Table 5-3 causes the database server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

Table 5-3. A Listen Thread for Each Listen Port

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1

The listen thread opens the port and requests one of the poll threads for the specified interface/protocol combination to monitor the port for client requests. The poll thread runs either in the CPU virtual processor or in the network virtual processor for the connection that is being used. For information on the number of poll threads, refer to "Specifying the Number of Networking Virtual Processors" on page 5-22.

For information on how to specify whether the poll threads for an interface/protocol combination run in CPU or network virtual processors, refer to "Running Poll Threads on CPU or Network Virtual Processors" on page 5-22 and to the NETTYPE configuration parameter in the *IBM Informix Administrator's Reference*.

When a poll thread receives a connection request from a client, it passes the request to the listen thread for the port. The listen thread authenticates the user, establishes the connection to the database server, and starts an **sqlexec** thread, the session thread that performs the primary processing for the client. Figure 5-8 illustrates the roles of the listen and poll threads in establishing a connection with a client application.

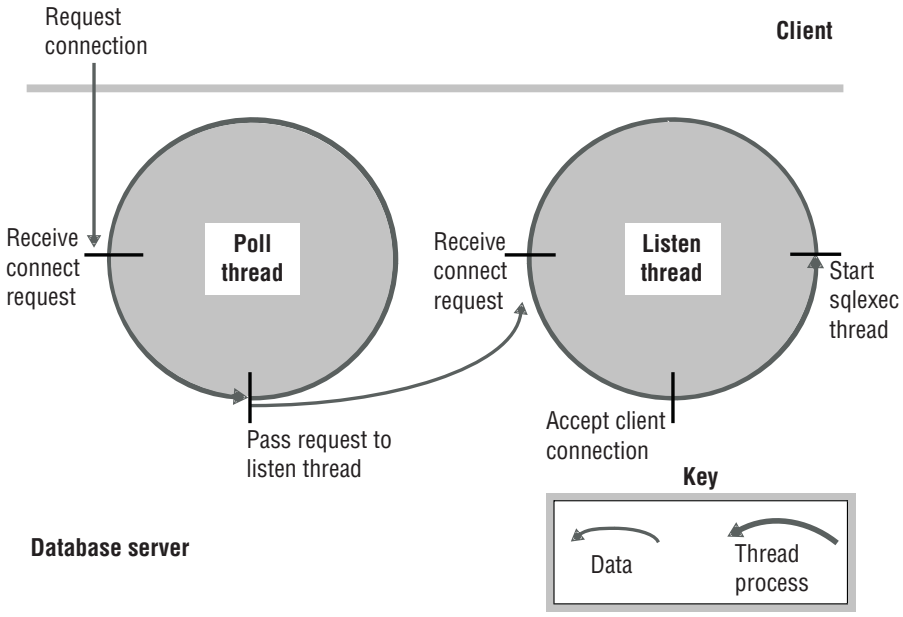


Figure 5-8. The Roles of the Poll and the Listen Threads in Connecting to a Client

A poll thread waits for requests from the client and places them in shared memory to be processed by the **sqlexec** thread. For network connections, the poll thread places the message in a queue in the shared-memory global pool. The poll thread then wakes up the **sqlexec** thread of the client to process the request. Whenever

possible, the **sqlexec** thread writes directly back to the client without the help of the poll thread. In general, the poll thread reads data from the client, and the **sqlexec** thread sends data to the client.

UNIX Only:

For a shared-memory connection, the poll thread places the message in the communications portion of shared memory.

Figure 5-9 illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

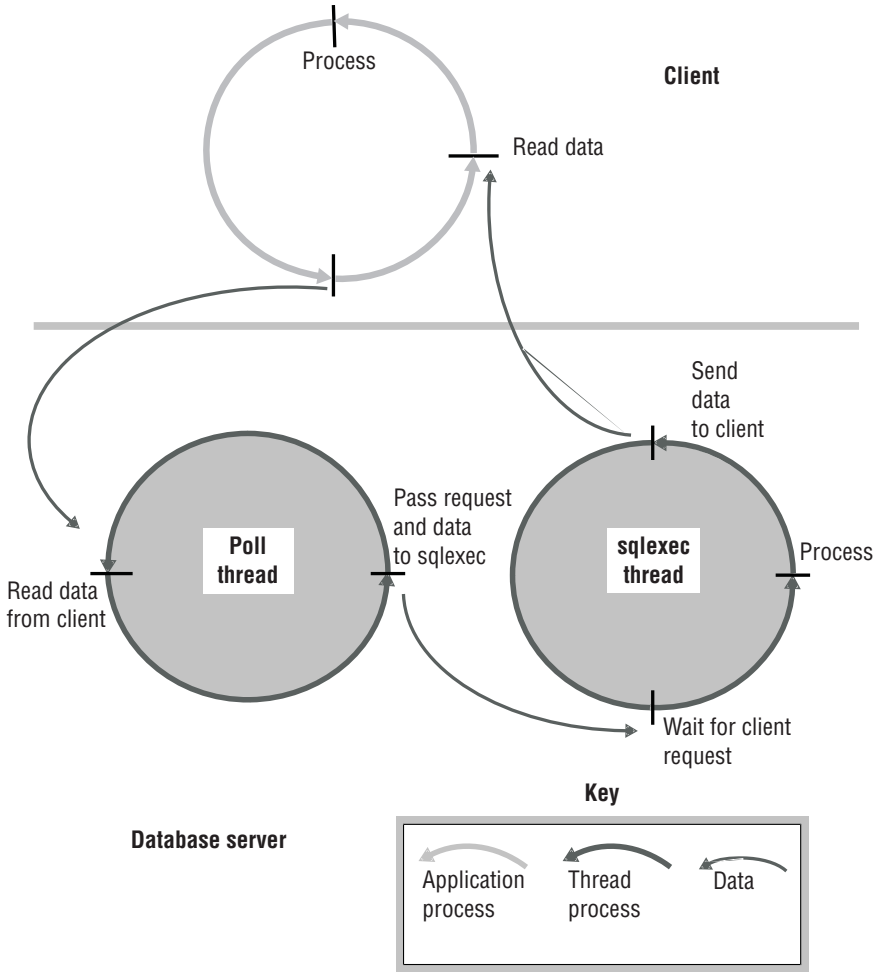


Figure 5-9. The Roles of the Poll and sqlexec Threads in Communicating with the Client Application

Using Fast Polling

You can use the FASTPOLL configuration parameter to enable or disable fast polling of your network, if your operating-system platform supports fast polling. Fast polling is beneficial if you have a large number of connections. For example, if you have more than 300 concurrent connections with the database server, you can enable the FASTPOLL configuration parameter for better performance.

To enable fast polling:

1. Set the FASTPOLL configuration parameter to 1.

If your operating system does not support fast polling, Dynamic Server ignores the FASTPOLL configuration parameter.

Starting Multiple Listen Threads

If the database server cannot service connection requests satisfactorily for a given interface/protocol combination with a single port and corresponding listen thread, you can improve service for connection requests in the following two ways:

- Add listen threads for additional ports.
- Add another network-interface card.

Adding Listen Threads:

As stated previously, the database server starts a listen thread for each dbservername that you specify with the DBSERVERNAME and DBSERVERALIASES configuration parameters.

To add listen threads for additional ports, you must first use the DBSERVERALIASES parameter to specify dbservernames for each of the ports. For example, the DBSERVERALIASES parameter in Figure 5-10 defines two additional dbservernames, **soc_ol2** and **soc_ol3**, for the database server instance identified as **soc_ol1**.

```
DBSERVERNAME      soc_ol1
DBSERVERALIASES   soc_ol2,soc_ol3
```

Figure 5-10. Defining Multiple dbservernames for Multiple Connections of the Same Type

After you define additional dbservernames for the database server, you must specify an interface/protocol combination and port for each of them in the **sqlhosts** file or registry. Each port is identified by a unique combination of **hostname** and **servicename** entries. For example, the **sqlhosts** entries shown in Table 5-4 cause the database server to start three listen threads for the **onsoctcp** interface/protocol combination, one for each of the ports defined.

Table 5-4. sqlhosts Entries to Listen to Multiple Ports for a Single Interface/Protocol Combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1
soc_ol2	onsoctcp	myhost	port2
soc_ol3	onsoctcp	myhost	port3

If you include a NETTYPE parameter for an interface/protocol combination, it applies to all the connections for that interface/protocol combination. In other words, if a NETTYPE parameter exists for **onsoctcp** in Table 5-4, it applies to all of the connections shown. In this example, the database server runs one *poll* thread for the **onsoctcp** interface/protocol combination unless the NETTYPE parameter specifies more. For more information about entries in the **sqlhosts** file or registry, refer to "Connectivity Files" on page 3-8.

Adding a Network-Interface Card:

If the network-interface card for the host computer cannot service connection requests satisfactorily, or if you want to connect the database server to more than one network, you can add a network-interface card.

To support multiple network-interface cards, you must assign each card a unique **hostname** (network address) in **sqlhosts**. For example, using the same dbservernames shown in Figure 5-10 on page 5-25, the **sqlhosts** file or registry entries shown in Table 5-5 cause the database server to start three listen threads for the same interface/protocol combination (as did the entries in Table 5-4 on page 5-25). In this case, however, two of the threads are listening to ports on one interface card (**myhost1**), and the third thread is listening to a port on the second interface card (**myhost2**).

Table 5-5. Example of sqlhosts Entries to Support Two Network-Interface Cards for the onsoctcp Interface/Protocol Combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost1	port1
soc_ol2	onsoctcp	myhost1	port2
soc_ol3	onsoctcp	myhost2	port1

Communications Support Module Virtual Processor

The communications support module (CSM) class of virtual processors performs communications support service and communications support module functions.

The database server starts the same number of CSM virtual processors as the number of CPU virtual processors that it starts.

For more information on the communications support service, refer to Chapter 3, "Client/Server Communications," on page 3-1.

Encrypt Virtual Processors

If the `encrypt` option of the VPCLASS parameter is not defined in the ONCONFIG configuration file, the database server starts one ENCRYPT VP the first time that any encryption or decryption functions defined for column-level encryption are called. You can define multiple ENCRYPT VPs if necessary to decrease the time needed to start the database server.

Use the VPCLASS configuration parameter with the `encrypt` keyword to configure encryption VPs. For example, to add five ENCRYPT VPs, add information in the ONCONFIG file as follows:

```
VPCLASS encrypt,num=5
```

You can modify the same information using the **onmode** utility, as follows:

```
onmode -p 5 encrypt
```

For more information, see the configuration parameters and the **onmode** utility chapter in the *IBM Informix Dynamic Server Administrator's Reference*. For more information on column-level encryption, see the *IBM Informix Security Guide*.

Optical Virtual Processor

The optical class (OPT) of virtual processors is used only with the Optical Subsystem. The Optical Subsystem starts one virtual processor in the optical class if the STAGEBLOB configuration parameter is present. For more information on the Optical Subsystem, see the *IBM Informix Optical Subsystem Guide*.

Audit Virtual Processor

The database server starts one virtual processor in the audit class (ADT) when you turn on audit mode by setting the ADTMODE parameter in the ONCONFIG file to 1. For more information about database server auditing, see the *IBM Informix Security Guide*.

Miscellaneous Virtual Processor

The miscellaneous virtual processor services requests for system calls that might require a very large stack, such as fetching information about the current user or the host-system name. Only one thread runs on this virtual processor; it executes with a stack of 128 kilobytes.

Chapter 6. Managing Virtual Processors

In This Chapter

This chapter describes how to set the configuration parameters that affect database server virtual processors, and how to start and stop virtual processors.

For descriptions of the virtual-processor classes and for advice on how many virtual processors you should specify for each class, refer to Chapter 5, "Virtual Processors and Threads," on page 5-1.

Setting Virtual-Processor Configuration Parameters

As user **root** or **informix**, use the following tools to set the configuration parameters for the database server virtual processors:

- A text editor
- IBM Informix Server Administrator (ISA)
- ON–Monitor

To implement any changes that you make to configuration parameters, you must shut down and restart the database server. For more information, refer to "Setting Up Shared Memory" on page 8-6.

Setting Virtual Processor Parameters with a Text Editor

You can use a text editor program to set ONCONFIG parameters at any time. Use the editor to locate the parameter that you want to change, enter the new value, and rewrite the file to disk.

Table 6-1 lists the ONCONFIG parameters that are used to configure virtual processors. For more information on how these parameters affect virtual processors, refer to "Virtual-Processor Classes" on page 5-11.

Table 6-1. Configuration Parameters for Configuring Virtual Processors

Parameter	Subparameters	Purpose
MULTIPROCESSOR		Specifies that you are running on a multiprocessor computer
NETTYPE		Specifies parameters for network protocol threads (and virtual processors)
SINGLE_CPU_VP		Specifies that you are running a single CPU virtual processor

VPCLASS	`adm kio shm`
	`adt lio soc`
	`aio msc str`
	`cpu ntk tli`
	`encrypt opt`
	`jvp pio`
		Specifies a predefined class name for the virtual processors. For example, `jvp` specifies a Java virtual processor (JVP).

VPCLASS	num=*number*	Specifies the number of virtual processors of the specified class that the database server should start

Table 6-1. Configuration Parameters for Configuring Virtual Processors (continued)

Parameter	Subparameters	Purpose
VPCLASS	max=*number*	Specifies the maximum number of virtual processors allowed for a class
VPCLASS	noage	Specifies the disabling of the aging priority
VPCLASS	aff = (*processor_number, start_range - end_range, start_range - end_range / increment*)	Specifies the assignment of virtual processors to CPUs, if processor affinity is available
VPCLASS	*user_defined*	Specifies user-defined virtual processor
VPCLASS	noyield	Specifies non-yielding virtual processor
VP_MEMORY_ CACHE_KB		Speeds access to memory blocks.

Specifying VPCLASS

You should use the VPCLASS parameter instead of the NUMCPUVPS, NUMAIOVPS, NOAGE, AFF_SPROC, and AFF_NPROCS parameters. You can specify a VPCLASS name of up to 128 bytes. The VPCLASS name must begin with a letter or underscore, and must contain letters, digits, underscores, or **$** characters.

You specify user-defined classes of virtual processors and Java virtual processors in the VPCLASS configuration parameter. For more information on the VPCLASS configuration parameter, including the defaults and value ranges, see the *IBM Informix Dynamic Server Administrator's Reference*, "User-Defined Classes of Virtual Processors" on page 5-15, and "Java Virtual Processors" on page 5-17.

Disabling Priority Aging (UNIX)

Use the VPCLASS parameter with the *noage* option to disable process priority aging on platforms that allow this feature.

For recommended values for these database server parameters on UNIX, refer to your machine notes file.

Setting Virtual-Processor Parameters with ISA

You can use ISA to display information about virtual processor classes, and monitor, add, or remove virtual-processor classes. For more information, see the ISA online help.

Setting Virtual-Processor Parameters with ON-Monitor

To set the virtual-processor configuration parameters with ON–Monitor, select the **Parameters > perFormance** option.

To specify network virtual processors, enter the number of virtual processors and then enter one of the following interface/protocol combinations: **ipcshm**, **ipcstr**, **tlitcp**, **tlispx**, or **soctcp**.

Starting and Stopping Virtual Processors

When you start the database server, the **oninit** utility starts the number and types of virtual processors that you have specified directly and indirectly. You configure virtual processors primarily through ONCONFIG parameters and, for network virtual processors, through parameters in the **sqlhosts** file or registry. For descriptions of the virtual-processor classes, refer to "Virtual-Processor Classes" on page 5-11.

The database server allows you to start a maximum of 1000 virtual processors.

After the database server is in online mode, you can start additional virtual processors to improve performance, if necessary. For more information, refer to "Adding Virtual Processors in Online Mode" below.

While the database server is in online mode, you can drop virtual processors of the CPU and user-defined classes. For more information, refer to "Dropping CPU and User-Defined Virtual Processors" on page 6-4.

To terminate the database server and thereby terminate all virtual processors, use the **onmode -k** command. For more information on using **onmode -k**, see the *IBM Informix Dynamic Server Administrator's Reference*.

Adding Virtual Processors in Online Mode

While the database server is in online mode, you can start additional virtual processors for the following classes: CPU, AIO, PIO, LIO, SHM, STR, TLI, SOC, JVP, and user-defined. The database server automatically starts one virtual processor each in the LIO and PIO classes unless mirroring is used, in which case it starts two.

You can start these additional virtual processors with one of the following methods:

- The **-p** option of the **onmode** utility
- ISA

You can also start additional virtual processors for user-defined classes to run user-defined routines. For more information about user-defined virtual processors, refer to "Assigning a UDR to a User-Defined Virtual-Processor Class" on page 5-16.

Adding Virtual Processors in Online Mode with onmode

Use the **-p** option of the **onmode** command to add virtual processors while the database server is in online mode. Specify the number of virtual processors that you want to add with a positive number. As an option, you can precede the number of virtual processors with a plus sign (+). Following the number, specify the virtual processor class in lowercase letters. For example, either of the following commands starts four additional virtual processors in the AIO class:

```
onmode -p 4 aio

onmode -p +4 aio
```

The **onmode** utility starts the additional virtual processors immediately.

You can add virtual processors to only one class at a time. To add virtual processors for another class, you must run **onmode** again.

Adding Virtual Processors in Online Mode with ON-Monitor (UNIX)

To use ON–Monitor to add virtual processors while the database server is online, select the **Modes > Add-Proc** option. You can add virtual processors in the following classes: CPU, AIO, LIO, PIO, and NET.

To specify network virtual processors, first enter the number of virtual processors and then enter one of the following interface/protocol combinations: **ipcshm**, **ipcstr**, **tlitcp**, **tlispx**, or **soctcp**.

You cannot use ON–Monitor to start additional virtual processors for a user-defined class. For more information, refer to "Adding Virtual Processors in Online Mode with onmode" on page 6-3.

Adding Network Virtual Processors

When you add network virtual processors, you are adding poll threads, each of which requires its own virtual processor to run. If you attempt to add poll threads for a protocol while the database server is in online mode, and you have specified in the NETTYPE configuration parameter that the poll threads run in the CPU class, the database server does not start the new poll threads if no CPU virtual processors are available to run them.

In the following example, the poll threads handle a total of 240 connections:

```
NETTYPE ipcshm,4,60,CPU # Configure poll thread(s) for nettype
```

For ipcshm, the number of poll threads correspond to the number of memory segments. For example, if NETTYPE is set to `3,100` and you want one poll thread, set the poll thread to `1,300`.

Dropping CPU and User-Defined Virtual Processors

While the database server is in online mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of the CPU and user-defined classes.

To drop CPU virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the CPU class in lowercase letters. For example, the following command drops two CPU virtual processors:

```
% onmode -p -2 cpu
```

If you attempt to drop a CPU virtual processor that is running a poll thread, you receive the following message:

```
onmode: failed when trying to change the number of cpu virtual processor by -number.
```

For more information, refer to "Running Poll Threads on CPU or Network Virtual Processors" on page 5-22.

To drop user-defined virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the user-defined class in lowercase letters. For example, the following command drops two virtual processors of the class *usr*:

```
onmode -p -2 usr
```

Windows Only:

In Windows, you can have only one user-defined virtual processor class at a time. Omit the *number* parameter in the **onmode -p *vpclass*** command.

For information on how to create a user-defined class of virtual processors and assign user-defined routines to it, refer to "User-Defined Classes of Virtual Processors" on page 5-15.

Monitoring Virtual Processors

Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity. For more information on these **onstat -g** options, see the chapter on the effect of configuration on CPU utilization in the *IBM Informix Dynamic Server Performance Guide*.

For examples of output for the **onstat -g** commands, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitoring Virtual Processors with Command-Line Utilities

You can use the following **onstat -g** options to monitor virtual processors:

- **ath**
- **glo**
- **ioq**
- **rea**

onstat -g ath

The **onstat -g ath** command displays information on system threads and the virtual-processor classes.

onstat -g glo

Use the **onstat -g glo** command to display information about each virtual processor that is currently running, as well as cumulative statistics for each virtual processor class. For an example of **onstat -g glo** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

onstat -g ioq

Use the **onstat -g ioq** option to determine whether you need to allocate additional virtual processors. The command **onstat -g ioq** displays the length and other statistics about I/O queues.

If the length of the I/O queue is growing, I/O requests are accumulating faster than the AIO virtual processors can process them. If the length of the I/O queue continues to show that I/O requests are accumulating, consider adding AIO virtual processors.

For an example of **onstat -g ioq** output, see information in the *IBM Informix Administrator's Reference*.

onstat -g rea

Use the **onstat -g rea** option to monitor the number of threads in the ready queue. If the number of threads in the ready queue is growing for a class of virtual processors (for example, the CPU class), you might need to add more virtual processors to your configuration.

For an example of **onstat -g rea** output, see information in the *IBM Informix Administrator's Reference*.

Monitoring Virtual Processors with SMI Tables

Query the **sysvpprof** table to obtain information on the virtual processors that are currently running. This table contains the following columns.

Column
Description

vpid Virtual-processor ID number

class Virtual-processor class

usercpu
Minutes of user CPU consumed

syscpu
Minutes of system CPU consumed

Chapter 7. Shared Memory

In This Chapter

This chapter describes the content of database server shared memory, the factors that determine the sizes of shared-memory areas, and how data moves into and out of shared memory. For information on how to change the database server configuration parameters that determine shared memory allocations, refer to Chapter 8, "Managing Shared Memory," on page 8-1.

Shared Memory

Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory. The database server uses shared memory for the following purposes:

- To reduce memory usage and disk I/O
- To perform high-speed communication between processes

Shared memory enables the database server to reduce overall memory usage because the participating processes, in this case, virtual processors, do not need to maintain private copies of the data that is in shared memory.

Shared memory reduces disk I/O, because buffers, which are managed as a common pool, are flushed on a database server-wide basis instead of a per-process basis. Furthermore, a virtual processor can often avoid reading data from disk because the data is already in shared memory as a result of an earlier read operation. The reduction in disk I/O reduces execution time.

Shared memory provides the fastest method of interprocess communication, because it processes read and write messages at the speed of memory transfers.

Shared-Memory Use

The database server uses shared memory for the following purposes:

- To enable virtual processors and utilities to share data
- To provide a fast communications channel for local client applications that use IPC communication

Figure 7-1 on page 7-2 illustrates the shared-memory scheme.

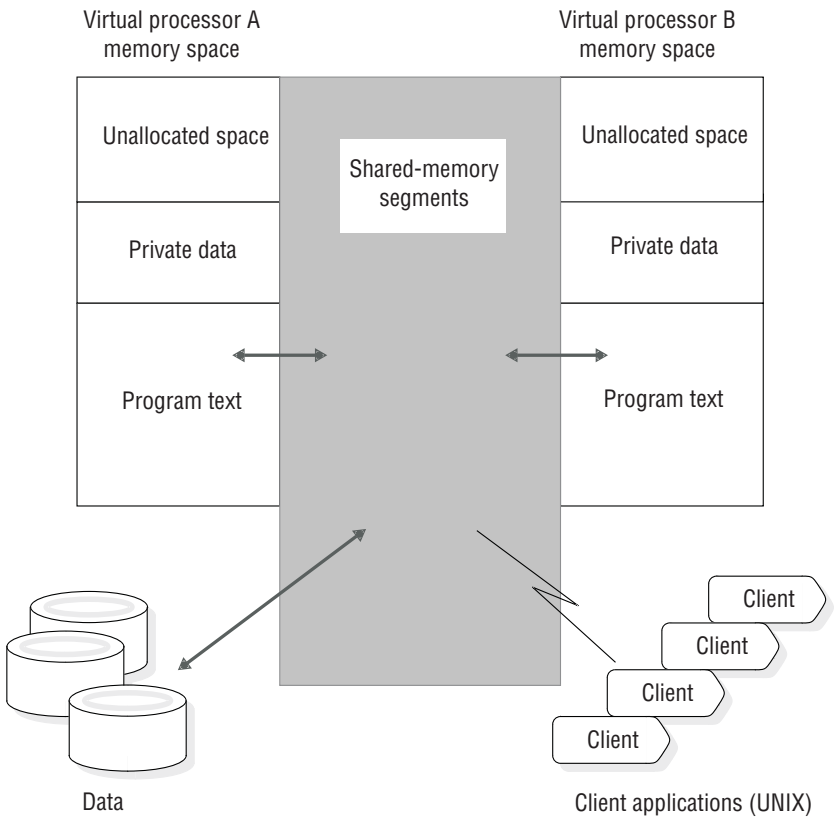


Figure 7-1. How the Database Server Uses Shared Memory

Shared-Memory Allocation

The database server creates the following portions of shared memory:

- The *resident* portion
- The *virtual* portion
- The IPC *communications* or message portion

  If the **sqlhosts** file specifies shared-memory communications, the database server allocates memory for the communications portion.
- The *virtual-extension* portion

The database server adds operating-system segments as needed to the virtual and virtual-extension portions of shared memory.

For more information on shared-memory settings for your platform, see the machine notes. Figure 7-2 on page 7-3 shows the contents of each portion of shared memory.

All database server virtual processors have access to the same shared-memory segments. Each virtual processor manages its work by maintaining its own set of pointers to shared-memory resources such as buffers, locks, and latches. Virtual processors attach to shared memory when you take the database server from offline mode to quiescent, administration, or online. The database server uses locks and latches to manage concurrent access to shared-memory resources by multiple

threads.

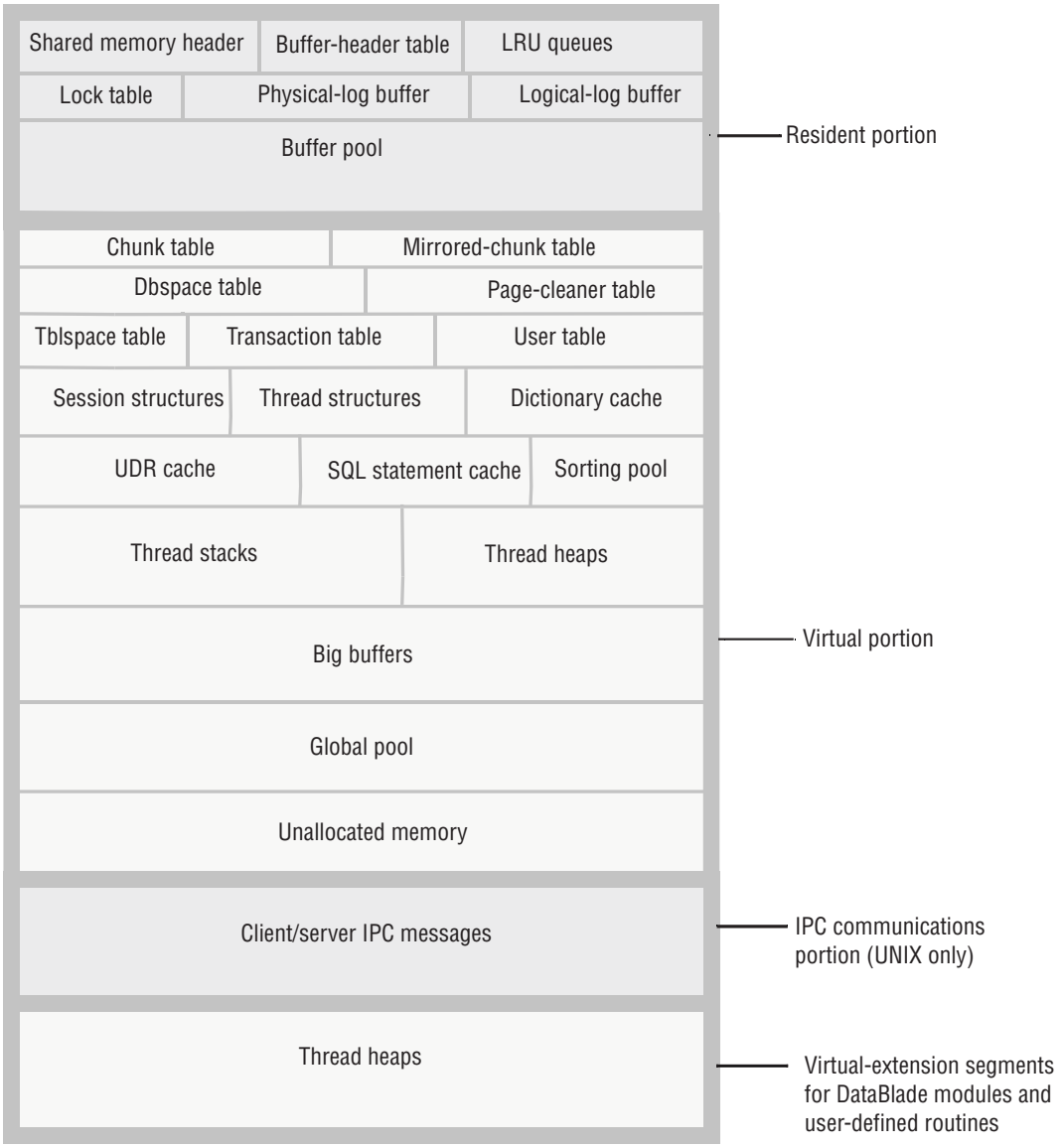


Figure 7-2. Contents of Database Server Shared Memory

Shared-Memory Size

Each portion of the database server shared memory consists of one or more operating-system segments of memory, each one divided into a series of blocks that are 4 kilobytes in size and managed by a bitmap.

The header-line output by the **onstat** utility contains the size of the database server shared memory, expressed in kilobytes. You can also use **onstat -g seg** to monitor how much memory the database server allocates for each portion of shared memory. For information on how to use **onstat**, see the *IBM Informix Dynamic Server Administrator's Reference*.

You can set the SHMTOTAL parameter in the ONCONFIG file to limit the amount of memory overhead that the database server can place on your computer or node. The SHMTOTAL parameter specifies the total amount of shared memory that the

database server can use for all memory allocations. However, certain operations might fail if the database server needs more memory than the amount set in SHMTOTAL. If this condition occurs, the database server displays the following message in the message log:

```
size of resident + virtual segments x + y > z
   total allowed by configuration parameter SHMTOTAL
```

In addition, the database server returns an error message to the application that initiated the offending operation. For example, if the database server needs more memory than you specify in SHMTOTAL while it tries to perform an operation such as an index build or a hash join, it returns an error message to the application that is similar to one of the following:

```
-567    Cannot write sorted rows.
-116    ISAM error: cannot allocate memory.
```

After the database server sends these messages, it rolls back any partial results performed by the offending query.

Internal operations, such as page-cleaner or checkpoint activity, can also cause the database server to exceed the SHMTOTAL ceiling. When this situation occurs, the database server sends a message to the message log. For example, suppose that the database server attempts and fails to allocate additional memory for page-cleaner activity. As a consequence, the database server sends a message to the message log that is similar to the following:

```
17:19:13          Assert Failed: WARNING! No memory available for page cleaners
17:19:13          Who: Thread(11, flush_sub(0), 9a8444, 1)
17:19:13          Results: Database server may be unable to complete a checkpoint
17:19:13          Action: Make more virtual memory available to database server
17:19:13          See Also: /tmp/af.c4
```

After the database server informs you about the failure to allocate additional memory, it rolls back the transactions that caused it to exceed the SHMTOTAL limit. Immediately after the rollback, operations no longer fail from lack of memory, and the database server continues to process transactions as usual.

Action to Take If SHMTOTAL Is Exceeded

When the database server needs more memory than SHMTOTAL allows, a transient condition occurs, perhaps caused by a burst of activity that exceeds the normal processing load. Only the operation that caused the database server to run out of memory temporarily should fail. Other operations continue to be processed in a normal fashion.

If messages indicate on a regular basis that the database server needs more memory than SHMTOTAL allows, you have not configured the database server correctly. Lowering DS_TOTAL_MEMORY or the **buffers** value in the BUFFERPOOL configuration parameter is one possible solution; increasing the value of SHMTOTAL is another.

Processes That Attach to Shared Memory

The following processes attach to the database server shared memory:

- Client-application processes that communicate with the database server through the shared-memory communications portion (**ipcshm**)
- Database server virtual processors
- Database server utilities

The following sections describe how each type of process attaches to the database server shared memory.

How a Client Attaches to the Communications Portion (UNIX)

Client-application processes that communicate with the database server through shared memory (nettype `ipcshm`) attach transparently to the communications portion of shared memory. System-library functions that are automatically compiled into the application enable it to attach to the communications portion of shared memory. For information on specifying a shared-memory connection, see Chapter 3, "Client/Server Communications," on page 3-1, and "Network Virtual Processors" on page 5-21.

If the **INFORMIXSHMBASE** environment variable is not set, the client application attaches to the communications portion at an address that is platform specific. If the client application attaches to other shared-memory segments (not database server shared memory), the user can set the **INFORMIXSHMBASE** environment variable to specify the address at which to attach the database server shared-memory communications segments. When you specify the address at which to address the shared-memory communications segments, you can prevent the database server from colliding with the other shared-memory segments that your application uses. For information on how to set the **INFORMIXSHMBASE** environment variable, refer to the *IBM Informix Guide to SQL: Reference*.

How Utilities Attach to Shared Memory

Database server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory through one of the following files.

Operating System	File
UNIX	**$INFORMIXDIR/etc/.infos.*servername***
Windows	**%INFORMIXDIR%\etc\.infos.*servername***

The variable **servername** is the value of the DBSERVERNAME parameter in the **ONCONFIG** file. The utilities obtain the **servername** portion of the filename from the **INFORMIXSERVER** environment variable.

The **oninit** process reads the ONCONFIG file and creates the file **.infos.servername** when it starts the database server. The file is removed when the database server terminates.

How Virtual Processors Attach to Shared Memory

The database server virtual processors attach to shared memory during setup. During this process, the database server must satisfy the following two requirements:

- Ensure that all virtual processors can locate and access the same shared-memory segments
- Ensure that the shared-memory segments reside in physical memory locations that are different than the shared-memory segments assigned to other instances of the database server, if any, on the same computer

The database server uses two configuration parameters, SERVERNUM and SHMBASE, to meet these requirements.

When a virtual processor attaches to shared memory, it performs the following major steps:

- Accesses the SERVERNUM parameter from the ONCONFIG file
- Uses SERVERNUM to calculate a shared-memory key value
- Requests a shared-memory segment using the shared-memory key value

  The operating system returns the shared-memory identifier for the first shared-memory segment.
- Directs the operating system to attach the first shared-memory segment to its process space at SHMBASE
- Attaches additional shared-memory segments, if required, to be contiguous with the first segment

The following sections describe how the database server uses the values of the SERVERNUM and SHMBASE configuration parameters in the process of attaching shared-memory segments.

Obtaining Key Values for Shared-Memory Segments

The values of the SERVERNUM configuration parameter and *shmkey*, an internally calculated number, determine the unique key value for each shared-memory segment.

To see the key values for shared-memory segments, run the **onstat -g seg** command. For more information, see the sections on SHMADD and the buffer pool in your *IBM Informix Dynamic Server Performance Guide*.

When a virtual processor requests that the operating system attach the first shared-memory segment, it supplies the unique key value to identify the segment. In return, the operating system passes back a *shared-memory segment identifier* associated with the key value. Using this identifier, the virtual processor requests that the operating system attach the segment of shared memory to the virtual-processor address space.

Specifying Where to Attach the First Shared-Memory Segment

The SHMBASE parameter in the ONCONFIG file specifies the virtual address where each virtual processor attaches the first, or base, shared-memory segment. Each virtual processor attaches to the first shared-memory segment at the same virtual address. This situation enables all virtual processors within the same database server instance to reference the same locations in shared memory without needing to calculate shared-memory addresses. All shared-memory addresses for an instance of the database server are relative to SHMBASE.

Warning: Do not change the value of SHMBASE.

The value of SHMBASE is sensitive for the following reasons:

- The specific value of SHMBASE depends on the platform and whether the processor is a 32-bit or 64-bit processor. The value of SHMBASE is not an arbitrary number and is intended to keep the shared-memory segments safe when the virtual processor dynamically acquires additional memory space.
- Different operating systems accommodate additional memory at different virtual addresses. Some architectures extend the highest virtual address of the virtual-processor data segment to accommodate the next segment. In this case, the data segment might grow into the shared-memory segment.

- Some versions of UNIX require the user to specify a SHMBASE parameter of virtual address zero. The zero address informs the UNIX kernel that the kernel should pick the best address at which to attach the shared-memory segments. However, not all UNIX architectures support this option. Moreover, on some systems, the selection that the kernel makes might not be the best selection.

For information on SHMBASE, see your Dynamic Server machine notes.

Attaching Additional Shared-Memory Segments

Each virtual processor must attach to the total amount of shared memory that the database server has acquired. After a virtual processor attaches each shared-memory segment, it calculates how much shared memory it has attached and how much remains. The database server facilitates this process by writing a shared-memory header to the first shared-memory segment. Sixteen bytes into the header, a virtual processor can obtain the following data:

- The total size of shared memory for this database server
- The size of each shared-memory segment

To attach additional shared-memory segments, a virtual processor requests them from the operating system in much the same way that it requested the first segment. For the additional segments, however, the virtual processor adds 1 to the previous value of *shmkey*. The virtual processor directs the operating system to attach the segment at the address that results from the following calculation:

```
SHMBASE + (seg_size x number of attached segments)
```

The virtual processor repeats this process until it has acquired the total amount of shared memory.

Given the initial key value of (`SERVERNUM * 65536`) + *shmkey*, the database server can request up to 65,536 shared-memory segments before it could request a shared-memory key value used by another database server instance on the same computer.

Defining the Shared-Memory Lower-Boundary Address

If your operating system uses a parameter to define the lower boundary address for shared memory, and the parameter is set incorrectly, it can prevent the shared-memory segments from being attached contiguously.

Figure 7-3 on page 7-8 illustrates the problem. If the lower-boundary address is less than the ending address of the previous segment plus the size of the current segment, the operating system attaches the current segment at a point beyond the end of the previous segment. This action creates a gap between the two segments. Because shared memory must be attached to a virtual processor so that it looks like contiguous memory, this gap creates problems. The database server receives errors when this situation occurs.

To correct the problem, check the operating-system kernel parameter that specifies the lower-boundary address or reconfigure the kernel to allow larger shared-memory segments.

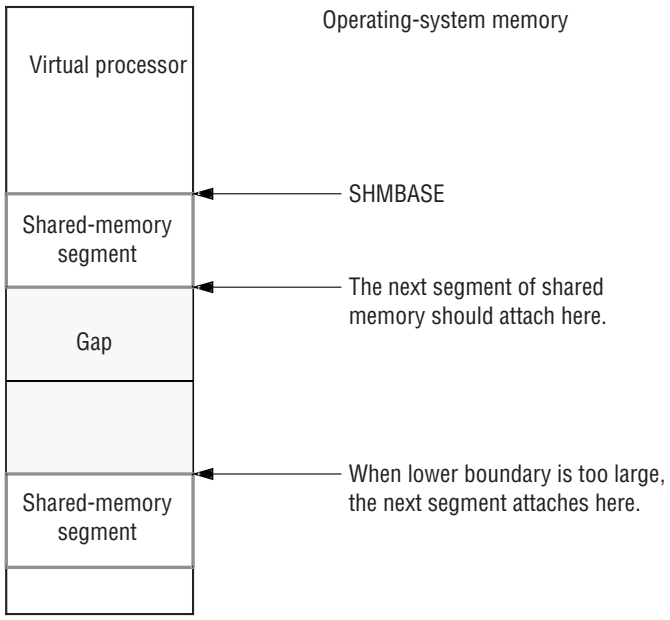


Figure 7-3. Shared-Memory Lower-Boundary Address Overview

Resident Shared-Memory Segments

The operating system, as it switches between the processes running on the system, normally swaps the contents of portions of memory to disk. When a portion of memory is designated as *resident*, however, it is not swapped to disk. Keeping frequently accessed data resident in memory improves performance because it reduces the number of disk I/O operations that would otherwise be required to access that data.

The database server requests that the operating system keep the virtual portions in physical memory when the following two conditions exist:

- The operating system supports shared-memory residency.
- The RESIDENT parameter in the ONCONFIG file is set to -1 or a value that is greater than 0.

Warning: You must consider the use of shared memory by all applications when you consider whether to set the RESIDENT parameter to *-1*. Locking all shared memory for the use of the Informix database server can adversely affect the performance of other applications, if any, on the same computer.

For more information on the RESIDENT configuration parameter, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

Resident Portion of Shared Memory

The resident portion of the database server shared memory stores the following data structures that do not change in size while the database server is running:

- Shared-memory header
- Buffer pool
- Logical-log buffer

- Physical-log buffer
- Lock table

Shared-Memory Header

The shared-memory header contains a description of all other structures in shared memory, including internal tables and the buffer pool.

The shared-memory header also contains pointers to the locations of these structures. When a virtual processor first attaches to shared memory, it reads address information in the shared-memory header for directions to all other structures.

The size of the shared-memory header is about 200 kilobytes, but the size varies depending on the computer platform. You cannot tune the size of the header.

Shared-Memory Buffer Pool

The buffer pool in the resident portion of shared memory contains buffers that store dbspace pages read from disk. The pool of buffers comprises the largest allocation of the resident portion of shared memory.

Figure 7-4 illustrates the shared-memory header and the buffer pool.

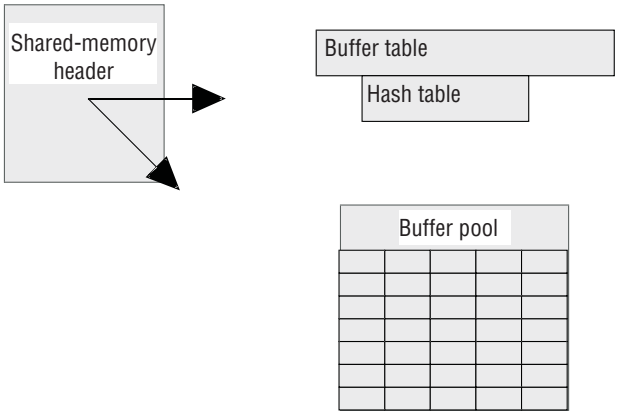


Figure 7-4. Shared-Memory Buffer Pool

You use the BUFFERPOOL configuration parameter to specify information about a buffer pool, including the number of buffers in the buffer pool. To allocate the proper number of buffers, start with at least four buffers per user. For more than 500 users, the minimum requirement is 2000 buffers. Too few buffers can severely impact performance, so you must monitor the database server and tune the value of buffers to determine an acceptable value. For more information on tuning the number of buffers, see the *IBM Informix Dynamic Server Performance Guide*.

If a buffer pool for a non-default page size does not exist, the database server will automatically create a large-page buffer.

If you are creating a dbspace with a non-default page size, the dbspace must have a corresponding buffer pool. For example, if you create a dbspace with a page size of 6 kilobytes, you must create a buffer pool with a size of 6 kilobytes.

Automatic LRU (least recently used) tuning affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

For more information on setting the BUFFERPOOL configuration parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

The status of the buffers is tracked through the buffer table. Within shared memory, buffers are organized into FIFO/LRU buffer queues. Buffer acquisition is managed through the use of latches, called *mutexes*, and lock-access information.

For a description of how LRU queues work, refer to "FIFO/LRU Queues" on page 7-21. For a description of mutexes, refer to "Mutexes" on page 5-11.

Buffer Overflow to the Virtual Portion

Because the maximum number of buffers in 64-bit addressing can be as large as 2^{31}-1, the resident portion of shared memory might not be able to hold all of the buffers in a large buffer pool. In this case, the virtual portion of database server shared memory might hold some of the buffers.

Buffer Size

Each buffer is the size of one database server page.

In general, the database server performs I/O in full-page units, the size of a buffer. The exceptions are I/O performed from big buffers, from blobspace buffers, or from lightweight I/O buffers. (See "Big Buffers" on page 7-17 and "Creation of Blobpage Buffers" on page 7-30.) For information about when to use private buffers, see information about lightweight I/O operations in the *IBM Informix Dynamic Server Performance Guide*.

The **onstat -b** command displays information about the buffers. For information on **onstat**, see the *IBM Informix Dynamic Server Administrator's Reference*.

Logical-Log Buffer

The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server. The logical log contains the following five types of log records, in addition to many others:

- SQL data definition statements for all databases
- SQL data manipulation statements for databases that were created with logging
- Record of a change to the logging status of a database
- Record of a checkpoint
- Record of a change to the configuration

The database server uses only one of the logical-log buffers at a time. This buffer is the current logical-log buffer. Before the database server flushes the current logical-log buffer to disk, it makes the second logical-log buffer the current one so that it can continue writing while the first buffer is flushed. If the second logical-log buffer fills before the first one finishes flushing, the third logical-log buffer becomes the current one. This process is illustrated in Figure 7-5 on page 7-11.

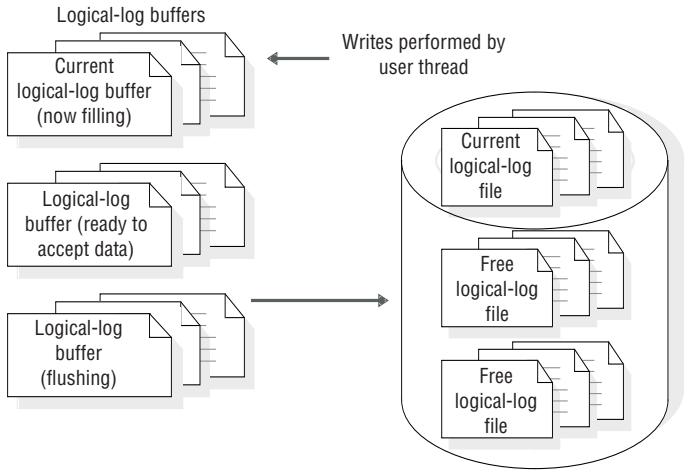


Figure 7-5. The Logical-Log Buffer and Its Relation to the Logical-Log Files on Disk

For a description of how the database server flushes the logical-log buffer, refer to "Flushing the Logical-Log Buffer" on page 7-28.

The LOGBUFF configuration parameter specifies the size of the logical-log buffers. Small buffers can create problems if you store records larger than the size of the buffers (for example, TEXT or BYTE data in dbspaces). The recommended value for the size of a logical log buffer is 64 kilobytes. Whenever the setting is less than the recommended value, the database server suggests a value during server startup. For the possible values that you can assign to this configuration parameter, see the *IBM Informix Dynamic Server Administrator's Reference*.

For information on the impact of TEXT and BYTE data on shared memory buffers, refer to "Buffering Large-Object Data" on page 7-29.

Physical-Log Buffer

The database server uses the physical-log buffer to hold before-images of some of the modified dbspace pages. The before-images in the physical log and the logical-log records enable the database server to restore consistency to its databases after a system failure.

The physical-log buffer is actually two buffers. Double buffering permits the database server processes to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. For a description of how the database server flushes the physical-log buffer, refer to "Flushing the Physical-Log Buffer" on page 7-26. For information on monitoring the physical-log file, refer to "Monitoring Physical and Logical-Logging Activity" on page 16-2.

The PHYSBUFF parameter in the ONCONFIG file specifies the size of the physical-log buffers. A write to the physical-log buffer writes exactly one page. If the specified size of the physical-log buffer is not evenly divisible by the page size, the database server rounds the size down to the nearest value that is evenly divisible by the page size. Although some operations require the buffer to be flushed sooner, in general the database server flushes the buffer to the physical-log file on disk when the buffer fills. Thus, the size of the buffer determines how frequently the database server needs to flush it to disk.

The default value for the physical log buffer size is 512 kilobytes. If you decide to use a smaller value, the database server displays a message indicating that optimal

performance might not be attained. Using a physical log buffer smaller than 512 kilobytes impacts performance only, not transaction integrity.

For more information on this configuration parameter, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

High-Availability Data-Replication Buffer

Data replication requires two instances of the database server, a primary instance and a secondary instance, running on two computers. If you implement data replication for your database server, the primary database server holds logical-log records in the data replication buffers before it sends them to the secondary database server. A data replication buffer is always the same size as the logical-log buffer. For information on the size of the logical-log buffer, refer to the preceding section, "Logical-Log Buffer" on page 7-10. For more information on how the data replication buffer is used, refer to "How Data Replication Works" on page 19-6.

Lock Table

A lock is created when a user thread writes an entry in the lock table. The lock table is the pool of available locks. A single transaction can own multiple locks. For an explanation of locking and the SQL statements associated with locking, see the *IBM Informix Guide to SQL: Tutorial*.

The following information, which is stored in the lock table, describes the lock:

- The address of the transaction that owns the lock
- The type of lock (exclusive, update, shared, byte, or intent)
- The page or rowid that is locked
- The table space where the lock is placed
- Information about the bytes locked (byte-range locks for smart large objects):
  - Smart-large-object ID
  - Offset into the smart large object where the locked bytes begin
  - The number of bytes locked, starting at the offset

To specify the initial size of the lock table, set the LOCKS configuration parameter. For information on using the LOCKS configuration parameter to specify the number of locks for a session, see the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference* and the chapter on configuration effects on memory utilization in your *IBM Informix Dynamic Server Performance Guide*.

If the number of locks allocated by sessions exceeds the value specified in the LOCKS configuration parameter, the database server doubles the size of the lock table, up to 15 times. The database server increases the size of the lock table by attempting to double the lock table on each increase. However, the amount added during each increase is limited to a maximum value. For 32-bit platforms, a maximum of 100,000 locks can be added during each increase. Therefore, the total maximum locks allowed for 32-bit platforms is 8,000,000 (maximum number of starting locks) + 99 (maximum number of dynamic lock table extensions) x 100,000 (maximum number of locks added per lock table extension). For 64-bit platforms, a maximum of 1,000,000 locks can be added during each increase. Therefore, the total maximum locks allowed is 500,000,000 (maximum number of starting locks) + 99 (maximum number of dynamic lock table extensions) x 1,000,000 (maximum number of locks added per lock table extension).

Use the DEF_TABLE_LOCKMODE configuration parameter to set the lock mode to page or row for new tables.

Locks can prevent sessions from reading data until after a concurrent transaction is committed or rolled back. For databases created with transaction logging, you can use the USELASTCOMMITTED configuration parameter in the ONCONFIG file to specify whether the database server uses the last committed version of the data. The last committed version of the data is the version of the data that existed before any updates occurred. The value you set with the USELASTCOMMITTED configuration parameter overrides the isolation level that is specified in the SET ISOLATION TO COMMITTED READ statement of SQL. For more information on using the USELASTCOMMITTED configuration parameter, see the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

For more information on using and monitoring locks, see the chapter on locking in your *IBM Informix Dynamic Server Performance Guide* and the *IBM Informix Guide to SQL: Tutorial*.

Virtual Portion of Shared Memory

The virtual portion of shared memory is expandable by the database server and can be paged out to disk by the operating system. As the database server executes, it automatically attaches additional operating-system segments, as needed, to the virtual portion.

Management of the Virtual Portion of Shared Memory

The database server uses memory *pools* to track memory allocations that are similar in type and size. Keeping related memory allocations in a pool helps to reduce memory fragmentation. It also enables the database server to free a large allocation of memory at one time, as opposed to freeing each piece that makes up the pool.

All sessions have one or more memory pools. When the database server needs memory, it looks first in the specified pool. If insufficient memory is available in a pool to satisfy a request, the database server adds memory from the system pool. If the database server cannot find enough memory in the system pool, it dynamically allocates more segments to the virtual portion.

The database server allocates virtual shared memory for each of its subsystems (session pools, stacks, heaps, control blocks, system catalog, SPL routine caches, SQL statement cache, sort pools, and message buffers) from pools that track free space through a linked list. When the database server allocates a portion of memory, it first searches the pool free-list for a *fragment* of sufficient size. If it finds none, it brings new blocks into the pool from the virtual portion. When memory is freed, it goes back to the pool as a free fragment and remains there until the pool is destroyed. When the database server starts a session for a client application, for example, it allocates memory for the session pool. When the session terminates, the database server returns the allocated memory as free fragments.

Size of the Virtual Portion of Shared Memory

To specify the initial size of the virtual shared-memory portion, set the SHMVIRTSIZE parameter in the ONCONFIG file. To specify the size of segments that are added later to the virtual shared memory, set the SHMADD or EXTSHMADD configuration parameter.

To specify the amount of memory available for PDQ queries, set the DS_TOTAL_MEMORY parameter.

If you want to increase the amount of memory that is available for a query that is not a PDQ query and the PDQ priority is set to 0 (zero), you can change the amount using any of the following options:

- The DS_NONPDQ_QUERY_MEM configuration parameter
- The **onmode -wm** or **onmode -wf** commands
- The **Non PDQ Query Memory** option on the ON-Monitor **pdQ** menu.

For example, if you use the **onmode** utility, specify a value as shown in the following example:

```
onmode -wf DS_NONPDQ_QUERY_MEM=500
```

The minimum value for DS_NONPDQ_QUERY_MEM is 128 kilobytes. The maximum supported value is 25 percent of the value of DS_TOTAL_MEMORY.

For more information on the SHMVIRTSIZE, SHMADD, EXTSHMADD, DS_TOTAL_MEMORY, DS_TOTAL_SIZE, and DS_NONPDQ_QUERY_MEM configuration parameters, see the *IBM Informix Dynamic Server Performance Guide* and the *IBM Informix Dynamic Server Administrator's Reference*. Also see "Adding a Segment to the Virtual Portion of Shared Memory" on page 8-7.

Components of the Virtual Portion of Shared Memory

The virtual portion of shared memory stores the following data:

- Internal tables
- Big buffers
- Session data
- Thread data (stacks and heaps)
- Data-distribution cache
- Dictionary cache
- SPL routine cache
- SQL statement cache
- Sorting pool
- Global pool

Shared-Memory Internal Tables

The database server shared memory contains seven internal tables that track shared-memory resources. The shared-memory internal tables are as follows:

- Buffer table
- Chunk table
- Dbspace table
- Page-cleaner table
- Tblspace table
- Transaction table
- User table

Buffer Table: The buffer table tracks the addresses and status of the individual buffers in the shared-memory pool. When a buffer is used, it contains an image of

a data or index page from disk. For more information on the purpose and content of a disk page, refer to "Pages" on page 9-4.

Each buffer in the buffer table contains the following control information, which is needed for buffer management:

- Buffer status

  Buffer status is described as empty, unmodified, or modified. An unmodified buffer contains data, but the data can be overwritten. A modified (dirty) buffer contains data that must be written to disk before it can be overwritten.
- Current lock-access level

  Buffers receive lock-access levels depending on the type of operation that the user thread is executing. The database server supports two buffer lock-access levels: shared and exclusive.
- Threads waiting for the buffer

  Each buffer header maintains a list of the threads that are waiting for the buffer and the lock-access level that each waiting thread requires.

Each database server buffer has one entry in the buffer table.

For information on the database server buffers, refer to "Resident Portion of Shared Memory" on page 7-8. For information on how to monitor the buffers, refer to "Monitoring Buffers" on page 8-8.

The database server determines the number of entries in the buffer-table hash table, based on the number of allocated buffers. The maximum number of hash values is the largest power of 2 that is less than the value of **buffers**, which is specified in one of the BUFFERPOOL configuration parameter fields.

Chunk Table: The chunk table tracks all chunks in the database server. If mirroring has been enabled, a corresponding mirror chunk table is also created when shared memory is set up. The mirror chunk table tracks all mirror chunks.

The chunk table in shared memory contains information that enables the database server to locate chunks on disk. This information includes the number of the initial chunk and the number of the next chunk in the dbspace. Flags also describe chunk status: mirror or primary; offline, online, or recovery mode; and whether this chunk is part of a blobspace. For information on monitoring chunks, refer to "Monitoring Chunks" on page 10-34.

The maximum number of entries in the chunk table might be limited by the maximum number of file descriptors that your operating system allows per process. You can usually specify the number of file descriptors per process with an operating-system kernel-configuration parameter. For details, consult your operating-system manuals.

Dbspace Table: The dbspace table tracks storage spaces in the database server. The dbspace-table information includes the following information about each dbspace:

- Dbspace number
- Dbspace name and owner
- Dbspace mirror status (mirrored or not)
- Date and time that the dbspace was created

If the storage space is a blobspace, flags indicate the media where the blobspace is located: magnetic, removable, or optical media. If the storage space is an sbspace, it contains internal tables that track metadata for smart large objects and large contiguous blocks of pages containing user data.

For information on monitoring dbspaces, refer to "Monitoring Disk Usage" on page 10-33.

Page-Cleaner Table: The page-cleaner table tracks the state and location of each of the page-cleaner threads. The number of page-cleaner threads is specified by the CLEANERS configuration parameter in the ONCONFIG file. For advice on how many page-cleaner threads to specify, refer to the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

The page-cleaner table always contains 128 entries, regardless of the number of page-cleaner threads specified by the CLEANERS parameter in the ONCONFIG file.

For information on monitoring the activity of page-cleaner threads, see information on the **onstat -F** option in the *IBM Informix Dynamic Server Administrator's Reference*.

Tblspace Table: The tblspace table tracks all active tblspaces in a database server instance. An active tblspace is one that is currently in use by a database session. Each active table accounts for one entry in the tblspace table. Active tblspaces include database tables, temporary tables, and internal control tables, such as system catalog tables. Each tblspace table entry includes header information about the tblspace, the tblspace name, and pointers to the tblspace **tblspace** in dbspaces on disk. (The shared-memory active tblspace table is different from the tblspace **tblspace**.) For information on monitoring tblspaces, refer to "Monitoring Tblspaces and Extents" on page 10-38.

The database server manages one tblspace table for each dbspace.

Transaction Table: The transaction table tracks all transactions in the database server.

Tracking information derived from the transaction table appears in the **onstat -x** display. For an example of the output that **onstat -x** displays, refer to monitoring transactions in your *IBM Informix Dynamic Server Performance Guide*.

The database server automatically increases the number of entries in the transaction table, up to a maximum of 32,767, based on the number of current transactions.

For more information on transactions and the SQL statements that you use with transactions, refer to the *IBM Informix Guide to SQL: Tutorial*, the *IBM Informix Guide to SQL: Reference*, and the *IBM Informix Guide to SQL: Syntax*.

UNIX Only:

The transaction table also specifically supports the X/Open environment. Support for the X/Open environment requires TP/XA. For a description of a transaction in this environment, refer to the *IBM Informix TP/XA Programmer's Manual*.

User Table: The user table tracks all user threads and system threads. Each client session has one primary thread and zero-to-many secondary threads, depending on

the level of parallelism specified. System threads include one to monitor and control checkpoints, one to process **onmode** commands, the B-tree scanner threads, and page-cleaner threads.

The database server increases the number of entries in the user table as needed. You can monitor user threads with the **onstat -u** command.

Big Buffers

A big buffer is a single buffer that is made up of several pages. The actual number of pages is platform dependent. The database server allocates big buffers to improve performance on large reads and writes.

The database server uses a big buffer whenever it writes to disk multiple pages that are physically contiguous. For example, the database server tries to use a big buffer to perform a series of sequential reads (light scans) or to read into shared memory simple large objects that are stored in a dbspace.

Users do not have control over the big buffers. If the database server uses light scans, it allocates big buffers from shared memory.

For information on monitoring big buffers with the **onstat** command, refer to the chapter on configuration effects on I/O activity in your *IBM Informix Dynamic Server Performance Guide*.

Session Data

When a client application requests a connection to the database server, the database server begins a *session* with the client and creates a data structure for the session in shared memory called the *session-control block*. The session-control block stores the session ID, the user ID, the process ID of the client, the name of the host computer, and various status flags.

The database server allocates memory for session data as needed.

Thread Data

When a client connects to the database server, in addition to starting a session, the database server starts a primary session thread and creates a *thread-control block* for it in shared memory.

The database server also starts internal threads on its own behalf and creates thread-control blocks for them. When the database server switches from running one thread to running another one (a context switch), it saves information about the thread— such as the register contents, program counter (address of the next instruction), and global pointers—in the thread-control block. For more information on the thread-control block and how it is used, refer to "Context Switching" on page 5-7.

The database server allocates memory for thread-control blocks as needed.

Stacks: Each thread in the database server has its own stack area in the virtual portion of shared memory. For a description of how threads use stacks, refer to "Stacks" on page 5-8. For information on how to monitor the size of the stack for a session, refer to monitoring sessions and threads section in your *IBM Informix Dynamic Server Performance Guide*.

The size of the stack space for user threads is specified by the STACKSIZE parameter in the ONCONFIG file. The default size of the stack is 32 kilobytes. You

can change the size of the stack for all user threads, if necessary, by changing the value of STACKSIZE. For information and a warning on setting the size of the stack, refer to STACKSIZE in the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

To alter the size of the stack for the primary thread of a specific session, set the **INFORMIXSTACKSIZE** environment variable. The value of **INFORMIXSTACKSIZE** overrides the value of STACKSIZE for a particular user. For information on how to override the stack size for a particular user, refer to the description of the **INFORMIXSTACKSIZE** environment variable in the *IBM Informix Guide to SQL: Reference*.

To more safely alter the size of stack space, use the **INFORMIXSTACKSIZE** environment variable rather than alter the configuration parameter STACKSIZE. The **INFORMIXSTACKSIZE** environment variable affects the stack space for only one user, and it is less likely to affect new client applications that initially were not measured.

Heaps: Each thread has a heap to hold data structures that it creates while it is running. A heap is dynamically allocated when the thread is created. The size of the thread heap is not configurable.

Data-Distribution Cache

The database server uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost. When the database server accesses the distribution statistics for a specific column the first time, it reads the distribution statistics from the **sysdistrib** system catalog table on disk and stores the statistics in the data-distribution cache. These statistics can then be read for the optimization of subsequent queries that access the column.

Performance improves if these statistics are efficiently stored and accessed from the data-distribution cache. You can configure the size of the data-distribution cache with the DS_HASHSIZE and DS_POOLSIZE configuration parameters. For information about changing the default size of the data-distribution cache, refer to the chapter on queries and the query optimizer in your *IBM Informix Dynamic Server Performance Guide*.

Dictionary Cache

When a session executes an SQL statement that requires access to a system catalog table, the database server reads data from the system catalog tables. The database server stores the catalog data for each queried table in structures that it can access more efficiently during subsequent queries on that table. These structures are created in the virtual portion of shared memory for use by all sessions. These structures constitute the dictionary cache.

You can configure the size of the dictionary cache with the DD_HASHSIZE and DD_HASHMAX configuration parameters. For more information about these parameters, refer to the chapter on configuration effects on memory in your *IBM Informix Dynamic Server Performance Guide*.

SQL Statement Cache

The SQL statement cache reduces memory usage and preparation time for queries. The database server uses the SQL statement cache to store parsed and optimized SQL statements that a user executes. When users execute a statement stored in the

SQL statement cache, the database server does not parse and optimize the statement again, so performance improves.

For more information, see "Setting SQL Statement Cache Parameters" on page 8-5. For details on how these parameters affect the performance of the SQL statement cache, refer to the *IBM Informix Dynamic Server Performance Guide*.

Sorting Memory

The following database operations can use large amounts of the virtual portion of shared memory to sort data:

- Decision-support queries that involve joins, groups, aggregates and sort operations
- Index builds
- UPDATE STATISTICS statement in SQL

The amount of virtual shared memory that the database server allocates for a sort depends on the number of rows to be sorted and the size of the row, along with other factors.

For information on parallel sorts, refer to your *IBM Informix Dynamic Server Performance Guide*.

SPL Routine and the UDR Cache

The database server converts an SPL routine to executable format and stores the routine in the UDR cache, where it can be accessed by any session.

When a session needs to access an SPL routine or other user-defined routine for the first time, the database server reads the definition from the system catalog tables and stores the definition in the UDR cache.

You can configure the size of the UDR cache with the PC_HASHSIZE and PC_POOLSIZE configuration parameters. For information about changing the default size of the UDR cache, refer to the chapter on queries and the query optimizer in your *IBM Informix Dynamic Server Performance Guide*.

Global Pool

The global pool stores structures that are global to the database server. For example, the global pool contains the message queues where poll threads for network communications deposit messages from clients. The **sqlexec** threads pick up the messages from the global pool and process them.

For more information, see the sections on network buffer pools and virtual portion of shared memory in your *IBM Informix Dynamic Server Performance Guide*.

Communications Portion of Shared Memory (UNIX)

The database server allocates memory for the IPC communication portion of shared memory if you configure at least one of your connections as an IPC shared-memory connection. The database server performs this allocation when you set up shared memory. The communications portion contains the message buffers for local client applications that use shared memory to communicate with the database server.

The size of the communications portion of shared memory equals approximately 12 kilobytes multiplied by the expected number of connections needed for

shared-memory communications (**nettype** ipcshm). If **nettype** ipcshm is not present, the expected number of connections defaults to 50. For information about how a client attaches to the communications portion of shared memory, refer to "How a Client Attaches to the Communications Portion (UNIX)" on page 7-5.

Virtual-Extension Portion of Shared Memory

The virtual-extension portion of shared memory contains additional virtual segments and virtual-extension segments. Virtual-extension segments contain thread heaps for DataBlade modules and user-defined routines that run in user-defined virtual processors.

The SHMADD, EXTSHMADD, and SHMTOTAL configuration parameters apply to the virtual-extension portion of shared memory, just as they do to the other portions of shared memory.

Concurrency Control

The database server threads that run on the same virtual processor and on separate virtual processors share access to resources in shared memory. When a thread writes to shared memory, it uses mechanisms called *mutexes* and *locks* to prevent other threads from simultaneously writing to the same area. A mutex gives a thread the right to access a shared-memory resource. A lock prevents other threads from writing to a buffer until the thread that placed the lock is finished with the buffer and releases the lock.

Shared-Memory Mutexes

The database server uses *mutexes* to coordinate threads as they attempt to modify data in shared memory. Every modifiable shared-memory resource is associated with a mutex. Before a thread can modify a shared-memory resource, it must first acquire the mutex associated with that resource. After the thread acquires the mutex, it can modify the resource. When the modification is complete, the thread releases the mutex.

If a thread tries to obtain a mutex and finds that it is held by another thread, the incoming thread must wait for the mutex to be released.

For example, two threads can attempt to access the same slot in the chunk table, but only one can acquire the mutex associated with the chunk table. Only the thread that holds the mutex can write its entry in the chunk table. The second thread must wait for the mutex to be released and then acquire it.

For information on monitoring mutexes (which are also referred to as latches), refer to "Monitoring the Shared-Memory Profile and Latches" on page 8-7.

Shared-Memory Buffer Locks

A primary benefit of shared memory is the ability of database server threads to share access to disk pages stored in the shared-memory buffer pool. The database server maintains thread isolation while it achieves this increased concurrency through a strategy for locking the data buffers.

Types of Buffer Locks

The database server uses two types of locks to manage access to shared-memory buffers:

- Share locks

- Exclusive locks

Each of these lock types enforces the required level of thread isolation during execution.

Share Lock: A buffer is in share mode, or has a share lock, if multiple threads have access to the buffer to read the data but none intends to modify the data.

Exclusive Lock: A buffer is in exclusive mode, or has an exclusive lock, if a thread demands exclusive access to the buffer. All other thread requests that access the buffer are placed in the wait queue. When the executing thread is ready to release the exclusive lock, it wakes the next thread in the wait queue.

Database Server Thread Access to Shared Buffers

Database server threads access shared buffers through a system of queues, using mutexes and locks to synchronize access and protect data.

FIFO/LRU Queues

A buffer holds data for the purpose of caching. The database server uses the least-recently used (LRU) queues to replace the cached data. Dynamic Server also has a first-in first-out (FIFO) queue. When you set the number of LRU queues, you are actually setting the number of FIFO/LRU queues.

Use the BUFFERPOOL configuration parameter to specify information about the buffer pool, including information about the number of LRU queues to create when database server shared memory is set up and values for **lru_min_dirty** and **lru_max_dirty**, which control how frequently the shared-memory buffers are flushed to disk.

To improve transaction throughput, increase the**lru_min_dirty** and **lru_max_dirty** values. However, do not change the gap between the **lru_min_dirty** and **lru_max_dirty** values.

Note: Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters prior to Version 10.0 is now specified using the BUFFERPOOL configuration parameter.

Components of LRU Queue

Each LRU queue is composed of a pair of linked lists, as follows:

- FLRU (free least-recently used) list, which tracks free or unmodified pages in the queue
- MLRU (modified least-recently used) list, which tracks modified pages in the queue

The free or unmodified page list is referred to as the FLRU queue of the queue pair, and the modified page list is referred to as the MLRU queue. The two separate lists eliminate the need to search a queue for a free or unmodified page. Figure 7-6 on page 7-22 illustrates the structure of the LRU queues.

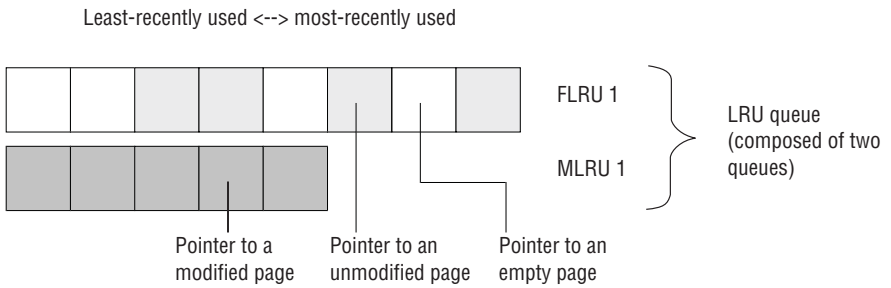


Figure 7-6. LRU Queue

Pages in Least-Recently Used Order

When the database server processes a request to read a page from disk, it must decide which page to replace in memory. Rather than select a page randomly, the database server assumes that recently referenced pages are more likely to be referenced in the future than pages that it has not referenced for some time. Thus, rather than replacing a recently accessed page, the database server replaces a least-recently accessed page. By maintaining pages in least-recently to most-recently used order, the database server can easily locate the least-recently used pages in memory.

LRU Queues and Buffer-Pool Management

Before processing begins, all page buffers are empty, and every buffer is represented by an entry in one of the FLRU queues. The buffers are evenly distributed among the FLRU queues. To calculate the number of buffers in each queue, divide the total number of buffers by the number of LRU queues. The number of buffers and LRU queues are specified in the BUFFERPOOL configuration parameter.

When a user thread needs to acquire a buffer, the database server randomly selects one of the FLRU queues and uses the oldest or least-recently used entry in the list. If the least-recently used page can be latched, that page is removed from the queue.

If the FLRU queue is locked, and the end page cannot be latched, the database server randomly selects another FLRU queue.

If a user thread is searching for a specific page in shared memory, it obtains the LRU-queue location of the page from the control information stored in the buffer table.

After an executing thread finishes its work, it releases the buffer. If the page has been modified, the buffer is placed at the most-recently used end of an MLRU queue. If the page was read but not modified, the buffer is returned to the FLRU queue at its most-recently used end. For information on how to monitor LRU queues, refer to "Monitoring Buffer-Pool Activity" on page 8-10.

Number of LRU Queues to Configure

Multiple LRU queues have two purposes:

- They reduce user-thread contention for the queues.
- They allow multiple cleaners to flush pages from LRU queues and maintain the percentage of dirty pages at an acceptable level.

Initial values for the LRUS are recommended based on the number of CPUs that are available on your computer. If your computer is a uniprocessor, start by setting the **lrus** value in the BUFFERPOOL configuration parameter to 4. If your computer is a multiprocessor, use the following formula:

```
LRUS = max(4, (NUMCPUVPS))
```

After you provide an initial value for **lrus** in the BUFFERPOOL configuration parameter, monitor your LRU queues with **onstat -R**. If you find that the percentage of dirty LRU queues consistently exceeds the value specified for **lru_max_dirty**, increase the value specified for **lrus** to add more LRU queues.

For example, suppose you set **lru_max_dirty** to 70 and find that your LRU queues are consistently 75 percent dirty. Consider increasing the value of the lrus. If you increase the number of LRU queues, you shorten the length of the queues, thereby reducing the work of the page cleaners. However, you must allocate a sufficient number of page cleaners with the CLEANERS configuration parameter, as discussed in the following section. Retain the same gap between **lru_max_dirty** and **lru_min_dirty**.

Note: Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters prior to Version 10.0 is now specified using the BUFFERPOOL configuration parameter.

Number of Cleaners to Allocate

In general, you should configure one cleaner for each disk that your applications update frequently. However, you should also consider the length of your LRU queues and frequency of checkpoints, as explained in the following paragraphs.

In addition to insufficient LRU queues, another factor that influences whether page cleaners keep up with the number of pages that require cleaning is whether you have enough page-cleaner threads allocated. The percent of dirty pages might exceed the BUFFERPOOL value specified for **lru_max_dirty** in some queues because no page cleaners are available to clean the queues. After a while, the page cleaners might be too far behind to catch up, and the buffer pool becomes dirtier than the percent that you specified in **lru_max_dirty**.

For example, suppose that the CLEANERS parameter is set to 8, and you increase the number of LRU queues from 8 to 12. You can expect little in the way of a performance gain because the 8 cleaners must now share the work of cleaning an additional 4 queues. If you increase the number of CLEANERS to 12, each of the now-shortened queues can be more efficiently cleaned by a single cleaner.

Setting CLEANERS too low can cause performance to suffer whenever a checkpoint occurs because page cleaners must flush all modified pages to disk during checkpoints. If you do not configure a sufficient number of page cleaners, checkpoints take longer, causing overall performance to suffer.

For more information, see “Flushing Buffer-Pool Buffers” on page 7-26.

Number of Pages Added to the MLRU Queues

Periodically, the page-cleaner threads flush the modified buffers in an MLRU queue to disk. To specify the point at which cleaning begins, use the BUFFERPOOL configuration parameter to specify a value for **lru_max_dirty**.

By specifying when page cleaning begins, the **lru_max_dirty** value limits the number of page buffers that can be appended to an MLRU queue. The initial

setting of **lru_max_dirty** is 60.00, so page cleaning begins when 60 percent of the buffers managed by a queue are modified.

In practice, page cleaning begins under several conditions, only one of which is when an MLRU queue reaches the value of **lru_max_dirty**. For more information on how the database server performs buffer-pool flushing, refer to "Flushing Data to Disk" on page 7-25.

Figure 7-7 shows how the value of **lru_max_dirty** is applied to an LRU queue to specify when page cleaning begins and thereby limit the number of buffers in an MLRU queue.

```
Buffers specified as 8000
lrus specified as 8
lru_max_dirty specified as 60 percent

Page cleaning begins when the number of buffers in the MLRU
   queue is equal to lru_max_dirty.

Buffers per lru queue = (8000/8) = 1000

Max buffers in MLRU queue and point at which page cleaning
   begins: 1000 x 0.60 = 600
```

Figure 7-7. How the lru_max_dirty Value Initiates Page Cleaning to Limit the Size of the MLRU Queue

End of MLRU Cleaning

You can also specify the point at which MLRU cleaning can end. The **lru_min_dirty** value in the BUFFERPOOL configuration parameter specifies the acceptable percentage of buffers in an MLRU queue. For example, if **lru_min_dirty** is set to 50.00, page cleaning is not required when 50 percent of the buffers in an LRU queue are modified. In practice, page cleaning can continue beyond this point, as directed by the page-cleaner threads.

Figure 7-8 shows how the value of **lru_min_dirty** is applied to the LRU queue to specify the acceptable percent of buffers in an MLRU queue and the point at which page cleaning ends.

```
Buffers specified as 8000
lrus specified as 8
lru_min_dirty specified as 50 percent

The acceptable number of buffers in the MLRU queue and
   the point at which page cleaning can end is equal
   to lru_min_dirty.

Buffers per LRU queue = (8000/8) = 1000

Acceptable number of buffers in MLRU queue and the point
   at which page cleaning can end: 1000 x .50 = 500
```

Figure 7-8. How the lru_min_dirty Value Specifies the Point at Which Page Cleaning Can End

You can use decimals for the **lru_max_dirty** and the **lru_min_dirty** values. For example, if you set **lru_max_dirty** to 1.0333 and **lru_min_dirty** to 1.0, this triggers the LRU to write at 3,100 dirty buffers and to stop at 3,000 dirty buffers.

For more information on how the database server flushes the buffer pool, refer to "Flushing Data to Disk" on page 7-25.

Configuring the Database Server to Read Ahead

For sequential table or index scans, you can configure the database server to read several pages ahead while the current pages are being processed. A read-ahead enables applications to run faster because they spend less time waiting for disk I/O.

The database server performs a read-ahead whenever it detects the need for it during sequential data or index reads.

The RA_PAGES parameter in the ONCONFIG file specifies the number of pages to read from disk or the index when the database server performs a read-ahead.

The RA_THRESHOLD parameter specifies the number of unprocessed pages in memory that cause the database server to perform another read-ahead. For example, if RA_PAGES is `10`, and RA_THRESHOLD is `4`, the database server reads ahead 10 pages when 4 pages remain to be processed in the buffer.

For more information on using the **onstat -p** command to monitor the database server use of read-ahead, see "Monitoring the Shared-Memory Profile and Latches" on page 8-7. For an example of **onstat -p** output, see the *IBM Informix Administrator's Reference*.

Database Server Thread Access to Buffer Pages

The database server uses shared-lock buffering to allow more than one database server thread to access the same buffer concurrently in shared memory. The database server uses two categories of buffer locks to provide this concurrency without a loss in thread isolation. The two categories of lock access are *share* and *exclusive*. (For more information, refer to "Types of Buffer Locks" on page 7-20.)

Flushing Data to Disk

Writing a buffer to disk is called *buffer flushing.* When a user thread modifies data in a buffer, it marks the buffer as *dirty*. When the database server flushes the buffer to disk, it subsequently marks the buffer as *not dirty* and allows the data in the buffer to be overwritten.

The database server flushes the following buffers:

- Buffer pool (covered in this section)
- Physical-log buffer

  See "Flushing the Physical-Log Buffer" on page 7-26.
- Logical-log buffer

  See "Flushing the Logical-Log Buffer" on page 7-28.

Page-cleaner threads manage buffer flushing. The database server always runs at least one page-cleaner thread. If the database server is configured for more than one page-cleaner thread, the LRU queues are divided among the page cleaners for more efficient flushing. For information on specifying how many page-cleaner threads the database server runs, refer to the CLEANERS configuration parameter in the *IBM Informix Dynamic Server Administrator's Reference*.

Flushing the physical-log buffer, the modified shared-memory page buffers, and the logical-log buffer must be synchronized with page-cleaner activity according to specific rules designed to maintain data consistency.

Flushing Buffer-Pool Buffers

Flushing of the buffers is initiated by any one of the following conditions:

- The number of buffers in an MLRU queue reaches the number specified by the **lru_max_dirty** value in the BUFFERPOOL configuration parameter.
- The page-cleaner threads cannot keep up. In other words, a user thread needs to acquire a buffer, but no unmodified buffers are available.
- The database server needs to execute a checkpoint. (See "Checkpoints" on page 15-4.)

Automatic LRU tuning affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

Flushing Before-Images First

The before-images of modified pages are flushed to disk before the modified pages themselves.

In practice, the physical-log buffer is flushed first and then the buffers that contain modified pages. Therefore, even when a shared-memory buffer page needs to be flushed because a user thread is trying to acquire a buffer but none is available (a foreground write), the buffer pages cannot be flushed until the before-image of the page has been written to disk.

Flushing the Physical-Log Buffer

The database server temporarily stores before-images of some of the modified disk pages in the physical-log buffer. If the before-image is written to the physical-log buffer but not to the physical log on disk, the server flushes the physical-log buffer to disk before flushing the modified page to disk.

The database server always flushes the contents of the physical-log buffer to disk before any data buffers.

The following events cause the active physical-log buffer to flush:

- The active physical-log buffer becomes full.
- A modified page in shared memory must be flushed, but the before-image is still in the active physical-log buffer.
- A checkpoint occurs.

The database server uses only one of the two physical-log buffers at a time. This buffer is the active (or current) physical-log buffer. Before the database server flushes the active physical-log buffer to disk, it makes the other buffer the active physical-log buffer so that the server can continue writing to a buffer while the first buffer is being flushed.

Both the physical-log buffer and the physical log help maintain the physical and logical consistency of the data. For information on physical logging, checkpoints, and fast recovery, refer to Chapter 15, "Physical Logging, Checkpoints, and Fast Recovery," on page 15-1.

Synchronizing Buffer Flushing

When shared memory is first set up, all buffers are empty. As processing occurs, data pages are read from disk into the buffers, and user threads begin to modify these pages.

Describing Flushing Activity

To provide you with information about the specific condition that prompted buffer-flushing activity, the database server defines three types of writes and counts how often each write occurs:

- Foreground write
- LRU write
- Chunk write

To display the write counts that the database server maintains, use onstat -F as described in the *IBM Informix Dynamic Server Administrator's Reference*.

If you implement mirroring for the database server, data is always written to the primary chunk first. The write is then repeated on the mirror chunk. Writes to a mirror chunk are included in the counts. For more information on monitoring the types of writes that the database server performs, refer to "Monitoring Buffer-Pool Activity" on page 8-10.

Foreground Write

Whenever an **sqlexec** thread writes a buffer to disk, it is termed a *foreground* write. A foreground write occurs when an **sqlexec** thread searches through the LRU queues on behalf of a user but cannot locate an empty or unmodified buffer. To make space, the **sqlexec** thread flushes pages, one at a time, to hold the data to be read from disk. (For more information, refer to "FIFO/LRU Queues" on page 7-21.)

If the **sqlexec** thread must perform buffer flushing just to acquire a shared-memory buffer, performance can suffer. Foreground writes should be avoided. To display a count of the number of foreground writes, run **onstat -F**. If you find that foreground writes are occurring on a regular basis, tune the value of the page-cleaning parameters. Either increase the number of page cleaners or decrease the BUFFERPOOL **lru_max_dirty** value.

LRU Write

Unlike foreground writes, LRU writes are performed by page cleaners rather than by **sqlexec** threads. The database server performs LRU writes as background writes that typically occur when the percentage of dirty buffers exceeds the percent that is specified for **lru_max_dirty** in the BUFFERPOOL configuration parameter.

In addition, a foreground write can trigger an LRU write. When a foreground write occurs, the **sqlexec** thread that performed the write alerts a page-cleaner to wake up and clean the LRU for which it performed the foreground write.

In a properly tuned system, page cleaners ensure that enough unmodified buffer pages are available for storing pages to be read from disk. Thus, **sqlexec** threads that perform a query do not need to flush a page to disk before they read in the disk pages required by the query. This condition can result in significant performance gains for queries that do not make use of foreground writes.

LRU writes are preferred over foreground writes because page-cleaner threads perform buffer writes much more efficiently than **sqlexec** threads do. To monitor both types of writes, use **onstat -F**.

Chunk Write

Chunk writes are commonly performed by page-cleaner threads during a checkpoint or, possibly, when every page in the shared-memory buffer pool is modified. Chunk writes, which are performed as sorted writes, are the most efficient writes available to the database server.

During a chunk write, each page-cleaner thread is assigned to one or more chunks. Each page-cleaner thread reads through the buffer headers and creates an array of pointers to pages that are associated with its specific chunk. (The page cleaners have access to this information because the chunk number is contained within the physical page number address, which is part of the page header.) This sorting minimizes head movement (disk seek time) on the disk and enables the page-cleaner threads to use the big buffers during the write, if possible.

In addition, because user threads must wait for the checkpoint to complete, the page-cleaner threads are not competing with a large number of threads for CPU time. As a result, the page-cleaner threads can finish their work with less context switching.

Flushing the Logical-Log Buffer

The database server uses the shared-memory logical-log buffer as temporary storage for records that describe modifications to database server pages. From the logical-log buffer, these records of changes are written to the current logical-log file on disk and eventually to the logical-log backup media. For a description of logical logging, refer to Chapter 13, "Logical Log," on page 13-1.

Five events cause the current logical-log buffer to flush:

- The current logical-log buffer becomes full.
- A transaction is prepared or committed in a database with unbuffered logging.
- A nonlogging database session terminates.
- A checkpoint occurs.
- A page is modified that does not require a before-image in the physical log.

The following sections discuss each of these events in detail.

After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging

The following log records cause flushing of the logical-log buffers in a database with unbuffered logging:

- COMMIT
- PREPARE
- XPREPARE
- ENDTRANS

For a comparison of buffered versus unbuffered logging, refer to the SET LOG statement in the *IBM Informix Guide to SQL: Syntax*.

When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates

Even for nonlogging databases, the database server logs certain activities that alter the database schema, such as the creation of tables or extents. When the database server terminates sessions that use unbuffered logging or nonlogging databases, the logical-log buffer is flushed to make sure that any logging activity is recorded.

When a Checkpoint Occurs

For a detailed description of the events that occur during a checkpoint, refer to "Checkpoints" on page 15-4.

When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File

When a page is modified that does not require a before-image in the physical log, the logical-log buffer must be flushed before that page is flushed to disk.

Buffering Large-Object Data

Simple large objects (TEXT or BYTE data) can be stored in either dbspaces or blobspaces. Smart large objects (CLOB or BLOB data) are stored only in sbspaces. The database server uses different methods to access each type of storage space. The following sections describe buffering methods for each.

Writing Simple Large Objects

The database server writes simple large objects to disk pages in a dbspace in the same way that it writes any other data type. For more information, refer to "Flushing Data to Disk" on page 7-25.

You can also assign simple large objects to a blobspace. The database server writes simple large objects to a blobspace differently from the way that it writes other data to a shared-memory buffer and then flushes it to disk. For a description of blobspaces, refer to the chapter on disk structure and storage in the *IBM Informix Dynamic Server Administrator's Reference*.

Blobpages and Shared Memory

Blobspace blobpages store large amounts of data. Consequently, the database server does not create or access blobpages by way of the shared-memory buffer pool, and it does not write blobspace blobpages to either the logical or physical logs.

If blobspace data passed through the shared-memory pool, it might dilute the effectiveness of the pool by driving out index pages and data pages. Instead, blobpage data is written directly to disk when it is created.

To reduce logical-log and physical-log traffic, the database server writes blobpages from magnetic media to dbspace backup tapes and logical-log backup tapes in a different way than it writes dbspace pages. For a description of how blobspaces are logged, refer to "Logging Blobspaces and Simple Large Objects" on page 13-6.

Blobpages stored on optical media are not written to dbspace and logical-log backup tapes due to the high reliability of optical media.

Creation of Simple Large Objects

When simple-large-object data is written to disk, the row to which it belongs might not exist yet. During an insert, for example, the simple large object is transferred before the rest of the row data. After the simple large object is stored, the data row is created with a 56-byte descriptor that points to its location. For a description of how simple large objects are stored physically, refer to the structure of a dbspace blobpage in the disk storage and structure chapter of the *IBM Informix Dynamic Server Administrator's Reference*.

Creation of Blobpage Buffers

To receive simple large object data from the application process, the database server creates a pair of blobspace buffers, one for reading and one for writing, each the size of one blobspace blobpage. Each user has only one set of blobspace buffers and, therefore, can access only one simple large object at a time.

Simple large object data is transferred from the client-application process to the database server in 1-kilobyte segments. The database server begins filling the blobspace buffers with the 1-kilobyte pieces and attempts to buffer two blobpages at a time. The database server buffers two blobpages so that it can determine when to add a forwarding pointer from one page to the next. When it fills the first buffer and discovers that more data remains to transfer, it adds a forward pointer to the next page before it writes the page to disk. When no more data remains to transfer, the database server writes the last page to disk without a forward pointer.

When the thread begins writing the first blobspace buffer to disk, it attempts to perform the I/O based on the user-defined blobpage size. For example, if the blobpage size is 32 kilobytes, the database server attempts to read or write the data in 32,768-byte increments. If the underlying hardware (such as the disk controller) cannot transfer this amount of data in a single operation, the operating-system kernel loops internally (in kernel mode) until the transfer is complete.

The blobspace buffers remain until the thread that created them is finished. When the simple large object is written to disk, the database server deallocates the pair of blobspace buffers. Figure 7-9 illustrates the process of writing a simple large object to a blobspace.

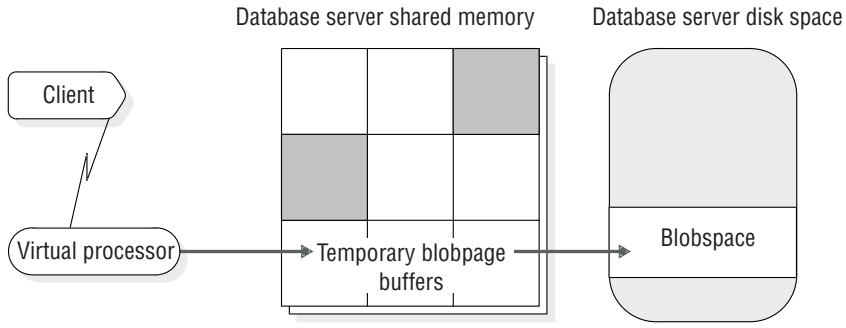


Figure 7-9. Writing Simple Large Object to a Blobspace

Blobspace blobpages are allocated and tracked with the free-map page. Links that connect the blobpages and pointers to the next blobpage segments are created as needed.

A record of the operation (insert, update, or delete) is written to the logical-log buffer.

Accessing Smart Large Objects

The database server accesses smart large objects through the shared-memory buffers, in the same way that it accesses data that is stored in a dbspace. However, the user-data portion of a smart large object is buffered at a lower priority than normal buffer pages to prevent flushing data of higher value out of the buffer pool. Buffering permits faster access to smart large objects that are accessed frequently.

A smart large object is stored in an sbspace. You cannot store simple large objects in an sbspace, and you cannot store smart large objects in a blobspace. An sbspace consists of a user-data area and a metadata area. The user-data area contains the smart-large-object data. The metadata area contains information about the content of the sbspace. For more information on sbspaces, refer to "Sbspaces" on page 9-12.

Because smart large objects pass through the shared-memory buffer pool and can be logged, you must consider them when you allocate buffers. Use the BUFFERPOOL configuration parameter to allocate shared-memory buffers. As a general rule, try to have enough buffers to hold two smart-large-object pages for each concurrently open smart large object. (The additional page is available for read-ahead purposes.) For more information about tuning buffers for smart large objects, refer to your *IBM Informix Dynamic Server Performance Guide*.

Use the LOGBUFF configuration parameter to specify the size of the logical-log buffer. For information about setting each of the following configuration parameters, refer to the *IBM Informix Dynamic Server Administrator's Reference*:

- BUFFERPOOL
- LOGBUFF

The user-data area of smart large objects that are logged does not pass through the physical log, so the PHYSBUFF parameter need not be adjusted for smart large objects.

For more information on the structure of an sbspace, refer to sbspace structure in the disk structures and storage chapter of the *IBM Informix Dynamic Server Administrator's Reference*. For information on creating an sbspace, see information on the **onspaces** utility in the *IBM Informix Dynamic Server Administrator's Reference*.

Memory Use on 64-Bit Platforms

With 64-bit addressing, you can have larger buffer pools to reduce the amount of I/O operations to obtain data from disks. Because 64-bit platforms allow for larger memory-address space, the maximum values for the following memory-related configuration parameters are larger on 64-bit platforms:

- BUFFERPOOL
- CLEANERS
- DS_MAX_QUERIES
- DS_TOTAL_MEMORY
- LOCKS
- LRUS

- SHMADD
- SHMVIRTSIZE

The machine notes for each 64-bit platform lists the maximum values for these configuration parameters and platform-specific parameters such as SHMMAX. For more information about the configuration parameters, see the *IBM Informix Dynamic Server Administrator's Reference* and the chapter on shared memory in the *IBM Informix Dynamic Server Performance Guide*.

Chapter 8. Managing Shared Memory

In This Chapter

This chapter tells you how to perform the following tasks, which concern managing shared memory:

- Setting the shared-memory configuration parameters
- Setting up shared memory
- Turning residency on or off for the resident portion of the database server shared memory
- Adding a segment to the virtual portion of shared memory
- Monitoring shared memory

This chapter does not cover the DS_TOTAL_MEMORY configuration parameter. This parameter places a ceiling on the allocation of memory for decision-support queries. For information on this parameter, refer to your *IBM Informix Dynamic Server Performance Guide*.

Setting Operating-System Shared-Memory Configuration Parameters

Several operating-system configuration parameters can affect the use of shared memory by the database server. Parameter names are not provided because names vary among platforms, and not all parameters exist on all platforms. The following list describes these parameters by function:

- Maximum operating-system shared-memory segment size, expressed in bytes or kilobytes
- Minimum shared-memory segment size, expressed in bytes
- Maximum number of shared-memory identifiers
- Lower-boundary address for shared memory
- Maximum number of attached shared-memory segments per process
- Maximum amount of systemwide shared memory

UNIX Only:

- Maximum number of semaphore identifiers
- Maximum number of semaphores
- Maximum number of semaphores per identifier

On UNIX, the machine notes file contains recommended values that you use to configure operating-system resources. Use these recommended values when you configure the operating system. For information on how to set these operating-system parameters, consult your operating-system manuals.

For specific information about your operating-system environment, refer to the machine notes file that is provided with the database server.

Maximum Shared-Memory Segment Size

When the database server creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible. The first

segment size that the database server tries to acquire is the size of the portion that it is allocating (resident, virtual, or communications), rounded up to the nearest multiple of 8 kilobytes.

The database server receives an error from the operating system if the requested segment size exceeds the maximum size allowed. If the database server receives an error, it divides the requested size by two and tries again. Attempts at acquisition continue until the largest segment size that is a multiple of 8 kilobytes can be created. Then the database server creates as many additional segments as it requires.

Using More Than Two Gigabytes of Memory (Windows)

The database server can access shared-memory segments larger than two gigabytes on Windows. However, you must enable this feature with an entry in the Windows boot file.

To add the entry, edit the boot.ini file (located in the top level, or root directory). You can either add a new boot option or use the currently existing boot option. To enable support for more than two gigabytes, add the following text to the end of the boot line:

```
/3GB
```

The following example has support for more than two gigabytes enabled:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows NT
Workstation Version 4.00"
/3GB
```

The maximum size of the shared-memory segment depends on the operating system, but it is approximately 3 gigabytes for Windows without additional drivers.

Maximum Number of Shared-Memory Identifiers (UNIX)

Shared-memory identifiers affect the database server operation when a virtual processor attempts to attach to shared memory. The operating system identifies each shared-memory segment with a shared-memory identifier. For most operating systems, virtual processors receive identifiers on a first-come, first-served basis, up to the limit that is defined for the operating system as a whole. For more information about shared-memory identifiers, refer to "How Virtual Processors Attach to Shared Memory" on page 7-5.

You might be able to calculate the maximum amount of shared memory that the operating system can allocate by multiplying the number of shared-memory identifiers by the maximum shared-memory segment size.

Semaphores (UNIX)

The database server operation requires one UNIX semaphore for each virtual processor, one for each user who connects to the database server through shared memory (**ipcshm** protocol), six for database server utilities, and sixteen for other purposes.

Setting Database Server Shared-Memory Configuration Parameters

Shared-memory configuration parameters fall into the following categories based on their purposes:

- Parameters that affect the resident portion of shared memory
- Parameters that affect the virtual portion of shared memory
- Parameters that affect performance

You can set shared-memory configuration parameters in the following ways:

- Using a text editor
- IBM Informix Server Administrator (ISA)
- Using ON–Monitor

On UNIX, you must be **root** or user **informix** to use either method. On Windows, you must be a user in the Informix Admin group.

Setting Parameters for Resident Shared Memory

Table 8-1 lists the parameters in the ONCONFIG file that specify the configuration of the buffer pool and the internal tables in the resident portion of shared memory. Before any changes that you make to the configuration parameters take effect, you must shut down and restart the database server. For a description of the configuration parameters, refer to the *IBM Informix Dynamic Server Administrator's Reference*.

Table 8-1. Configuring the Resident Portion of Shared Memory

Configuration Parameter	Purpose
BUFFERPOOL	Specifies information about the buffer pool that must be defined for each different page size that a dbspace uses.
LOCKS	Specifies the initial number of locks for database objects; for example, rows, key values, pages, and tables.
LOGBUFF	Specifies the size of the logical-log buffers.
PHYSBUFF	Specifies the size of the physical-log buffers.
RESIDENT	Specifies residency for the resident portion of the database server shared memory.
SERVERNUM	Specifies a unique identification number for the database server on the local host computer.
SHMTOTAL	Specifies the total amount of memory to be used by the database server.

Setting Parameters for Virtual Shared Memory

Table 8-2 lists the ONCONFIG parameters that you use to configure the virtual portion of shared memory. For more information, see the chapter on configuration effects on memory in your *IBM Informix Dynamic Server Performance Guide*.

Table 8-2. Configuring the Virtual Portion of Shared Memory

Configuration Parameter	Purpose
DS_HASHSIZE	Number of hash buckets for lists in the data-distribution cache.
DS_POOLSIZE	Maximum number of entries in the data-distribution cache.

Table 8-2. Configuring the Virtual Portion of Shared Memory (continued)

Configuration Parameter	Purpose
PC_HASHSIZE	Specifies the number of hash buckets for the UDR cache and other caches that the database server uses. For more information about setting PC_HASHSIZE, refer to your *IBM Informix Dynamic Server Performance Guide*.
PC_POOLSIZE	Specifies the number of UDRs (SPL routines and external routines) that can be stored in the UDR cache. In addition, this parameter specifies the size of other database server caches, such as the typename cache and the opclass cache. For more information about setting PC_POOLSIZE, refer to your *IBM Informix Dynamic Server Performance Guide*.
SHMADD	Specifies the size of dynamically added shared-memory segments.
SHMNOACCES	Specifies a list of virtual memory address ranges that will not be used to attach shared memory. Use this parameter to avoid conflicts with other processes.
EXTSHMADD	Specifies the size of an added extension segment.
SHMTOTAL	Specifies the total amount of memory to be used by the database server.
SHMVIRTSIZE	Specifies the initial size of the virtual portion of shared memory.
STACKSIZE	Specifies the stack size for the database server user threads.

Setting Parameters for Shared-Memory Performance

Table 8-3 lists the ONCONFIG parameters that set shared-memory performance options. For more information, see the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

Table 8-3. Setting Shared-Memory Performance Options

Configuration Parameter	Purpose
CKPTINTVL	Specifies the maximum number of seconds that can elapse before the database server checks if a checkpoint is needed and the RTO_SERVER_RESTART configuration parameter is not set to turn on automatic checkpoint tuning.
CLEANERS	Specifies the number of page-cleaner threads that the database server is to run.
RA_PAGES	Specifies the number of disk pages that the database server should attempt to read ahead when it performs sequential scans of data or index records.
RA_THRESHOLD	Specifies the number of unprocessed memory pages that, after they are read, cause the database server to read ahead on disk.

Setting Shared-Memory Parameters with a Text Editor

You can use a text editor to set the configuration parameters for resident and virtual shared memory, and shared-memory performance. Locate the parameter in the ONCONFIG file, enter the new value or values, and rewrite the file to disk. Before the changes take effect, however, you must shut down and restart the database server.

Setting Shared-Memory Parameters with ISA

Use ISA to monitor and set the following shared-memory parameters. For more information, see the ISA online help:

- Execute utility commands such as **onmode** and **onstat**.
- Edit ONCONFIG parameters.

- Monitor segments
- Monitor pools
- Monitor resident memory
- Monitor nonresident memory
- Monitor data dictionary cache

Setting Shared-Memory Parameters with ON-Monitor (UNIX)

To set the configuration parameters for the resident and virtual portions of shared memory with ON–Monitor, select the **Parameters > Shared-Memory** option.

To determine the page size for your system, choose the **Parameters > Shared-memory** option in ON–Monitor. The database server page size is the last entry on the page.

Important: The configuration parameters SHMADD, EXTSHMADD, and SHMTOTAL affect both the resident and virtual portions of shared memory.

To set the configuration parameters for the following shared-memory performance options with ON–Monitor, select the **Parameters > perFormance** option:

- CKPTINTVL
- RA_PAGES
- RA_THRESHOLD

Setting SQL Statement Cache Parameters

Table 8-4 shows the different ways that you can configure the SQL statement cache.

Table 8-4. Configuring the SQL Statement Cache

Configuration Parameter	Purpose	onmode Command
STMT_CACHE	Turns on, enables, or disables the SQL statement cache in memory. If turned on, specifies whether the SQL statement cache can hold a parsed and optimized SQL statement.	**onmode -e** *mode*
STMT_CACHE_HITS	Specifies the number of hits (references) to a statement before it is fully inserted into the SQL statement cache.	**onmode -W STMT_CACHE_HITS**
STMT_CACHE_NOLIMIT	Controls whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.	**onmode -W STMT_CACHE_NOLIMIT**
STMT_CACHE_NUMPOOL	Defines the number of memory pools for the SQL statement cache.	None
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.	None

Use the following **onstat** options to monitor the SQL statement cache:

- **onstat -g ssc** (same as **onstat -g cac stmt**)
- **onstat -g ssc all**
- **onstat -g ssc pool**

For more information on these configuration parameters, **onstat -g** options, and **onmode** commands, see the *IBM Informix Dynamic Server Administrator's Reference*.

For more information on using the SQL statement cache, monitoring it with the **onstat -g** options, and tuning the configuration parameters, see improving query performance in the *IBM Informix Dynamic Server Performance Guide*. For details on qualifying and identical statements, see the *IBM Informix Guide to SQL: Syntax*.

Setting Up Shared Memory

To set up shared memory, take the database server offline and then online. For information on how to take the database server from online mode to offline, refer to "Change From Any Mode Immediately to Offline Mode" on page 4-14.

Turning Residency On or Off for Resident Shared Memory

You can turn residency on or off for the resident portion of shared memory in either of the following two ways:

- Use the **onmode** utility to reverse the state of shared-memory residency immediately while the database server is in online mode.
- Change the RESIDENT parameter in the ONCONFIG file to turn shared-memory residency on or off for the next time that you set up the database server shared memory.

For a description of the resident portion of shared memory, refer to "Resident Portion of Shared Memory" on page 7-8.

Turning Residency On or Off in Online Mode

To turn residency on or off while the database server is in online mode, use the **onmode** utility.

To turn on residency immediately for the resident portion of shared memory, execute the following command:

```
% onmode -r
```

To turn off residency immediately for the resident portion of shared memory, execute the following command:

```
% onmode -n
```

These commands do not change the value of the RESIDENT parameter in the ONCONFIG file. That is, this change is not permanent, and residency reverts to the state specified by the RESIDENT parameter the next time that you set up shared memory. On UNIX, you must be **root** or user **informix** to turn residency on or off. On Windows, you must be a user in the Informix Admin group to turn residency on or off.

Turning Residency On or Off When Restarting the Database Server

You can use a text editor to turn residency on or off. To change the current state of residency, use a text editor to locate the RESIDENT parameter. Set RESIDENT to 1 to turn residency on or to 0 to turn residency off, and rewrite the file to disk. Before the changes take effect, you must shut down and restart the database server.

Adding a Segment to the Virtual Portion of Shared Memory

The **-a** option of the **onmode** utility allows you to add a segment of specified size to virtual shared memory.

You do not normally need to add segments to virtual shared memory because the database server automatically adds segments as needed.

The option to add a segment with the **onmode** utility is useful if the number of operating-system segments is limited, and the initial segment size is so low, relative to the amount that is required, that the operating-system limit of shared-memory segments is nearly exceeded.

Monitoring Shared Memory

This section describes how to monitor shared-memory segments, the shared-memory profile, and the use of specific shared-memory resources (buffers, latches, and locks).

You can use the **onstat -o** utility to capture a static snapshot of database server shared memory for later analysis and comparison.

Monitoring Shared-Memory Segments

Monitor the shared-memory segments to determine the number and size of the segments that the database server creates. The database server allocates shared-memory segments dynamically, so these numbers can change. If the database server is allocating too many shared-memory segments, you can increase the SHMVIRTSIZE configuration parameter. For more information, see the chapter on configuration parameters in the *IBM Informix Dynamic Server Administrator's Reference*.

The **onstat -g seg** command lists information for each shared-memory segment, including the address and size of the segment, and the amount of memory that is free or in use. For an example of **onstat -g seg** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitoring the Shared-Memory Profile and Latches

Monitor the database server profile to analyze performance and the use of shared-memory resources. The Profile screen maintains cumulative statistics on shared-memory use. To reset these statistics to zero, use the **onstat -z** option. For a description of all the fields that **onstat** displays, see information on the **onstat** utility in the *IBM Informix Dynamic Server Administrator's Reference*.

You can obtain statistics on latch use and information on specific latches. These statistics provide a measure of the system activity.

Using Command-Line Utilities

You can use the following command-line utilities to monitor shared memory and latches:

- **onstat -s**
- **onstat -p**

onstat -s:

Use **onstat -s** command to obtain latch information.

onstat -p:

Execute **onstat -p** to display statistics on database server activity and waiting latches (in the **lchwaits** field). For an example of **onstat -p** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

Using IBM Informix Server Administrator

You can use ISA to obtain information about latches, spin locks, and profiles.

Using ON-Monitor (UNIX)

Select **Status > Profile**. The screen displays shared-memory statistics, as well as the current operating mode, the boot time, the current time, and latches.

Using SMI Tables

Query the **sysprofile** table to obtain shared-memory statistics. This table contains all of the statistics available in **onstat -p** output except the **ovbuff**, **usercpu**, and **syscpu** statistics.

Monitoring Buffers

You can obtain both statistics on buffer use and information on specific buffers. The statistical information includes the percentage of data writes that are cached to buffers and the number of times that threads had to wait to obtain a buffer. The percentage of writes cached is an important measure of performance. (For information on how to use this statistic to tune the database server, see your *IBM Informix Dynamic Server Performance Guide*.)

The number of waits for buffers gives a measure of system concurrency.

Information on specific buffers includes a listing of all the buffers in shared memory that are held by a thread. This information allows you to track the status of a particular buffer. For example, you can determine if another thread is waiting for the buffer.

Using Command-Line Utilities

You can use the following command-line utilities to monitor buffers:

- **onstat -p**
- **onstat -B**
- **onstat -b**
- **onstat -X**
- **onstat -R**

onstat -p:

Execute **onstat -p** to obtain statistics about cached reads and writes. The following caching statistics appear in four fields on the top row of the output display:

- The number of reads from shared-memory buffers (**bufreads**)

- The percentage of reads cached (**%cached**)
- The number of writes to shared memory (**bufwrits**)
- The percentage of writes cached (**%cached**)
- Information on generic pages (nonstandard pages in the buffer pool)

In the output, the number of reads or writes can appear as a negative number if the number of occurrences exceeds 2^{32} (depends on the platform).

The **onstat -p** option also displays a statistic (**bufwaits**) that indicates the number of times that sessions had to wait for a buffer.

For an example of **onstat -p** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

onstat -B:

Execute **onstat -B** to obtain information on all buffers currently in use, including:

- The address of every regular shared-memory buffer
- The address of the thread that currently holds the buffer
- The address of the first thread that is waiting for each buffer
- Information on buffer pools

For an example of **onstat -B** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

onstat -b:

Execute **onstat -b** to obtain the following information about each buffer:

- Address of each buffer currently held by a thread
- Page numbers for the page held in the buffer
- Type of page held in the buffer (for example, data page, tblspace page, and so on)
- Type of lock placed on the buffer (exclusive or shared)
- Address of the thread that is currently holding the buffer
- Address of the first thread that is waiting for each buffer
- Information on buffer pools

You can compare the addresses of the user threads to the addresses that appear in the **onstat -u** display to obtain the session ID number.

For more information on the fields that **onstat** displays, see information on the **onstat** utility in the *IBM Informix Dynamic Server Administrator's Reference*

onstat -X:

Execute **onstat -X** to obtain the same information as for **onstat -b**, along with the *complete* list of all threads that are waiting for buffers, not just the first waiting thread.

onstat -R:

Use **onstat -R** to display information about buffer pools, including information about buffers.

Using ON-Monitor (UNIX)

To access the fields mentioned on page "onstat -p" on page 8-8 for **onstat -p** (**bufreads**, **%cached**, **bufwrits %cached**), select the **Status > Profile** option.

Here is an example of cached read and write statistics in the **Profile** option of the ON-Monitor **Status** menu:

```
...

Disk Reads  Buff. Reads    %Cached  Disk Writes  Buff. Writes    %Cached
       177          330      46.36            4             0        0.00
...
```

Using SMI Tables

Query the **sysprofile** table to obtain statistics on cached reads and writes and total buffer waits. The following rows are relevant.

Row Description

dskreads
Number of reads from disk

bufreads
Number of reads from buffers

dskwrites
Number of writes to disk

bufwrites
Number of writes to buffers

buffwts
Number of times that any thread had to wait for a buffer

Monitoring Buffer-Pool Activity

You can obtain statistics that relate to buffer availability as well as information on the buffers in each LRU queue.

The statistical information includes the number of times that the database server attempted to exceed the maximum number of buffers and the number of writes to disk (categorized by the event that caused the buffers to flush). These statistics help you determine if the number of buffers is appropriate. For information on tuning database server buffers, see your *IBM Informix Dynamic Server Performance Guide*.

Information on the buffers in each LRU queue consists of the length of the queue and the percentage of the buffers in the queue that have been modified.

Using Command-Line Utilities

You can use the **onstat** utility to obtain information on buffer-pool activity. You also can execute **onstat** options from ISA.

For more information about the **onstat** options, see information on the **onstat** utility in the *IBM Informix Dynamic Server Administrator's Reference*.

onstat -p:

The **onstat -p** output contains a statistic (**ovbuff**) that indicates the number of times the database server attempted to exceed the maximum number of shared buffers specified by `buffers` value in the BUFFERPOOL configuration parameter.

onstat -F:

Execute **onstat -F** to obtain a count by write type of the writes performed. (For an explanation of the different write types, see "Describing Flushing Activity" on page 7-27.

The **onstat -F** command displays totals for the following write types:

- Foreground write
- LRU write
- Chunk write

The **onstat -F** command also lists the following information about the page cleaners:

- Page-cleaner number
- Page-cleaner shared-memory address
- Current state of the page cleaner
- LRU queue to which the page cleaner was assigned

For an example of **onstat -F** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

onstat -R:

Execute **onstat -R** to obtain information about the number of buffers in each LRU queue and the number and percentage of the buffers that are modified or free.

For an example of **onstat -R** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*

Using SMI Tables

Query the **sysprofile** table to obtain the statistics on write types that are held in the following rows.

Row Description

fgwrites
Number of foreground writes

lruwrites
Number of LRU writes

chunkwrites
Number of chunk writes

Deleting Shared Memory Segments After a Server Failure

You should close shared memory segments after a database server failure.

Warning: This procedure should be performed by a DBA with experience using IDS. Please consult technical support for assistance. This procedure is for UNIX systems only.

In the event of a failure of an IDS database server instance, follow this procedure to delete shared memory segments:

1. Log on as user **informix**.
2. Use the **onmode -k** command to take the database server to offline mode and remove shared memory.
3. If the **onmode -k** command fails and the server is not off line, either run the + **onclean -k** command, or perform the following steps:
   a. Use the **onstat -g glo** command to display multithreading information.
   b. In the output from the above command, find the process ID (pid) associated with the first instance of **cpu** in the class column. For example, in the following output from the **onstat -g glo** command, there are four occurrences of **cpu** in the class column, having pids of 2599, 2603, 2604, and 2605:

```
MT global info:
sessions threads  vps      lngspins
0        49       14       1
          sched calls     thread switches yield 0   yield n   yield forever
total:    900100          898846          1238      27763     423778
per sec:  327             325             2         12        151
Virtual processor summary:
 class       vps       usercpu   syscpu    total
 cpu         4         0.92      0.10      1.02
 aio         4         0.02      0.02      0.04
 lio         1         0.00      0.00      0.00
 pio         1         0.00      0.00      0.00
 adm         1         0.00      0.01      0.01
 msc         1         0.00      0.00      0.00
 fifo        2         0.00      0.00      0.00
 total       14        0.94      0.13      1.07
Individual virtual processors:
 vp    pid       class       usercpu   syscpu    total
 1     2599      cpu         0.25      0.06      0.31
 2     2602      adm         0.00      0.01      0.01
 3     2603      cpu         0.23      0.00      0.23
 4     2604      cpu         0.21      0.03      0.24
 5     2605      cpu         0.23      0.01      0.24
 6     2606      lio         0.00      0.00      0.00
 7     2607      pio         0.00      0.00      0.00
 8     2608      aio         0.02      0.02      0.04
 9     2609      msc         0.00      0.00      0.00
 10    2610      fifo        0.00      0.00      0.00
 11    2611      fifo        0.00      0.00      0.00
 12    2612      aio         0.00      0.00      0.00
 13    2613      aio         0.00      0.00      0.00
 14    2614      aio         0.00      0.00      0.00
                 tot         0.94      0.13      1.07
```

   c. Use the **kill** command to terminate (in order) process IDs 2599, 2603, 2604, and 2605.
4. If the shared segments have not been removed then follow these steps:
   a. Determine the server number. The server number can be found by examining the **ONCONFIG** file of the IDS instance
   b. Add the server number to 21078. For example, if the server number is 1, then add 1 to 21078, giving 21079.
   c. Convert the sum from the previous step to hexadecimal. In the previous example, 21079 is 5257 hexadecimal.
   d. Concatenate 48 to the hex value from the previous step. For example, 525748.

e. Run the **ipcs** utility as root to display the shared memory segments, if any, left open by the server. Search the key column for the number from step d.

f. Remove each shared memory ID associated with the number from step d.

+ For more information on the **onclean** utility, see the *IBM Informix Dynamic Server*
+ *Administrator's Reference*.

Consult your operating system documentation for the correct **ipcm** syntax for your system.

Chapter 9. Data Storage

In This Chapter

This chapter defines terms and explains the concepts that you must understand to perform the tasks described in Chapter 10, "Managing Disk Space," on page 10-1. This chapter covers the following topics:

- Definitions of the physical and logical units that the database server uses to store data on disk
- Instructions on how to calculate the amount of disk space that you need to store your data
- Guidelines on how to lay out your disk space and where to place your databases and tables

See the current Dynamic Server release notes for any supplementary information on the maximum values related to the storage units discussed in this chapter.

Physical and Logical Units of Storage

The database server uses the physical units of storage to allocate disk space. Unlike the logical units of storage whose size fluctuates, each of the physical units has a fixed or assigned size that is determined by the disk architecture. The database server uses the following physical units to manage disk space:

- Chunk
- Page
- Extent
- Blobpage
- Sbpage

The database server stores data in the following logical units:

- Dbspace
- Temporary dbspace
- Blobspace
- Sbspace
- Temporary sbspace
- Extspace
- Database
- Table
- Tblspace
- Partition

The database server maintains the following storage structures to ensure physical and logical consistency of data:

- Logical log
- Physical log
- Reserved pages

The following sections describe the various data-storage units that the database server supports and the relationships between those units. For information about reserved pages, see the disk structures and storage chapter in the *IBM Informix Dynamic Server Administrator's Reference*.

Chunks

A *chunk* is the largest unit of physical disk dedicated to database server data storage. Chunks provide administrators with a significantly large unit for allocating disk space. The maximum size of an individual chunk is 4 terabytes. The number of allowable chunks is 32,766. You must run onmode -BC to enable the maximum size of a chunk and the maximum number allowable. If onmode -BC is not run, then the maximum chunk size is 2 gigabytes.

The database server administrator adds one or more chunks to the following storage spaces when they approach full capacity:

- Dbspace (see page "Dbspaces" on page 9-8)
- Blobspace (see page "Blobspaces" on page 9-11)
- Sbspace (see page "Sbspaces" on page 9-12)

For information on chunk names, size, and number, see "Specifying Names for Storage Spaces and Chunks" on page 10-5 and "Specifying the Maximum Size of Chunks" on page 10-6.

The database server also uses chunks for mirroring. A *primary chunk* is a chunk from which the database server copies data to a *mirror chunk*. If the primary chunk fails, the database server brings the mirror chunk online automatically. For more information, see Chapter 17, "Mirroring," on page 17-1.

Disk Allocation for Chunks

The database server can use regular operating-system files or *raw disk devices* to store data. On UNIX, you should use raw disk devices to store data whenever performance is important. On Windows, using NTFS files to store data is recommended for ease of administration.

An Informix storage space can reside on an NFS-mounted file system using regular operating-system files.

Disk Access on Windows

On Windows, both raw disks and NTFS use kernel asynchronous I/O (KAIO). The Windows file system manager adds additional overhead to disk I/O, so using raw disks provides slight performance advantages. Because NTFS files are a more standard method of storing data, you should use NTFS files instead of raw disks. Consider using raw disks if your database server requires a large amount of disk access.

Raw Disk Space on Windows: On Windows, *raw disk space* can be either a physical drive without a drive letter or a logical disk partition that has been assigned a drive letter using the **Disk Administrator**. The space can either be formatted or unformatted. If it contains data, the data will be overwritten after the space has been allocated to the database server. For more information, see "Allocating Raw Disk Space on Windows" on page 10-5.

NTFS Files: You must use NTFS files, not FAT files, for disk space on Windows. For more information, see "Allocating NTFS File Space on Windows" on page 10-4.

Unbuffered or Buffered Disk Access on UNIX

You can allocate disk space in two ways. You can either use files that are buffered through the operating system, or you can use unbuffered disk access.

Files that are buffered through the operating system are often called *cooked* files.

Unbuffered disk access is also called *raw* disk space.

When dbspaces reside on *raw disk devices* (also called *character-special devices*), the database server uses unbuffered disk access.

To create a raw device, configure a *block device* (hard disk) with a raw interface. The storage space that the device provides is called *raw disk space*. A chunk of raw disk space is physically contiguous.

The name of the chunk is the name of the character-special file in the **/dev** directory. In many operating systems, you can distinguish the character-special file from the block-special file by the first letter in the filename (typically *r*). For example, **/dev/rsd0f** is the character-special device that corresponds to the **/dev/sd0f** block-special device.

For more information, see "Allocating Raw Disk Space on UNIX" on page 10-3.

A *cooked file* is a regular file that the operating system manages. Cooked file chunks and raw disk chunks are equally reliable. Unlike raw disk space, the logically contiguous blocks of a cooked file might not be physically contiguous.

You can more easily allocate cooked files than raw disk space. To allocate a cooked file, you need only create the file on any existing partition. The name of the chunk is the complete path name of the file. These steps are described in "Allocating Cooked File Spaces on UNIX" on page 10-3.

In a learning environment, where performance is not critical, or for static data, cooked files can be convenient. If you must use cooked UNIX files, store the least frequently accessed data in those files. Store the files in a file system with minimal activity.

For cooked file chunks, the operating system processes all chunk *I/O* from its own buffer pool and ensures that all writes to chunks are physically written to the disk.

Important: While you should generally use raw disk devices on UNIX to achieve better performance, if you enable the DIRECT_IO configuration parameter, the performance for cooked files can approach the performance of raw devices used for dbspace chunks. This occurs because direct I/O bypasses the use of the file system buffers. If you have an AIX operating system, you can also enable concurrent I/O for Dynamic Server to use with direct IO when reading and writing to chunks that use cooked files. For more information on using direct IO or concurrent IO, see the *IBM Informix Dynamic Server Performance Guide*.

To determine the best device for performance, perform benchmark testing on the system with both types of devices for the dbspace and table layout.

When using raw disks, you do not need to take any special action to create chunks and files that are larger than two gigabytes. If you want to create large chunks in

cooked files, or if you want to use the various database export and import utilities with large files, you must ensure that the files systems that will hold the large files are properly configured.

Offsets

The system administrator might divide a physical disk into *partitions,* which are different parts of a disk that have separate path names. Although you should use an entire disk partition when you allocate a chunk on a raw disk device, you can subdivide partitions or cooked files into smaller chunks using *offsets.* For more information, see "Disk-Layout Guidelines" on page 9-33.

Tip: With a 4-terabyte limit to the size of a chunk, you can avoid partitioning a disk by assigning a single chunk per disk drive.

An offset allows you to indicate the location of a given chunk on the disk partition, file, or device. For example, suppose that you create a 1000-kilobyte chunk that you want to divide into two chunks of 500 kilobytes each. You can use an offset of zero kilobytes to mark the beginning of the first chunk and an offset of 500 kilobytes to mark the beginning of the second chunk.

You can specify an offset whenever you create, add, or drop a chunk from a dbspace, blobspace, or sbspace.

You might also need to specify an offset to prevent the database server from overwriting partition information. "Allocating Raw Disk Space on UNIX" on page 10-3 explains when and how to specify an offset.

Pages

A *page* is the physical unit of disk storage that the database server uses to read from and write to Informix databases. Figure 9-1 illustrates the concept of a page, represented by a darkened sector of a disk platter.

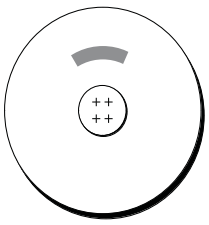


Figure 9-1. A Page on Disk

On most UNIX platforms, the page size is 2 kilobytes. On Windows, the page size is 4 kilobytes. Because your hardware determines the size of your page, you cannot alter this value.

A chunk contains a certain number of pages, as Figure 9-2 on page 9-5 illustrates. A page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.

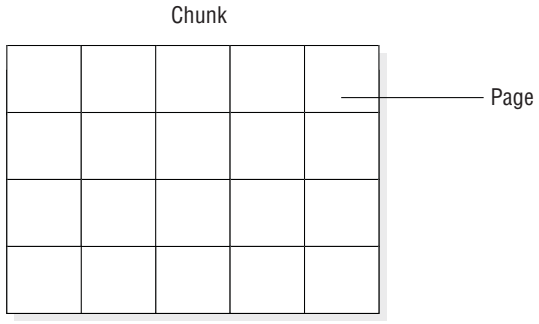


Figure 9-2. A Chunk, Logically Separated into a Series of Pages

For information on how the database server structures data within a page, see the chapter on disk structures and storage in the *IBM Informix Dynamic Server Administrator's Reference*

Blobpages

A blobpage is the unit of disk-space allocation that the database server uses to store simple large objects (TEXT or BYTE data) within a blobspace. For a description of blobspaces, refer to "Blobspaces" on page 9-11.

You specify blobpage size as a multiple of the database server page size. Because the database server allocates blobpages as contiguous spaces, it is more efficient to store simple large objects in blobpages that are as close to the size of the data as possible. Figure 9-3 illustrates the concept of a blobpage, represented as a multiple (three) of a data page.

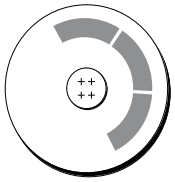


Figure 9-3. A Blobpage on Disk

For information about how Dynamic Server structures data stored in a blobpage, see structure of a blobspace blobpage in the disk structures and storage chapter of the *IBM Informix Dynamic Server Administrator's Reference*.

Just as with pages in a chunk, a certain number of blobpages compose a chunk in a blobspace, as Figure 9-4 on page 9-6 illustrates. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

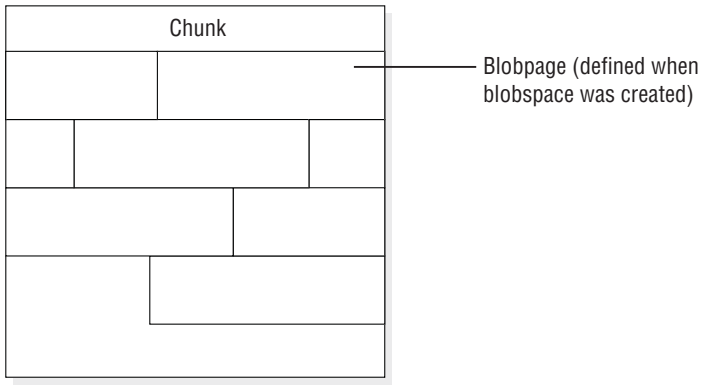


Figure 9-4. A Chunk in a Blobspace, Logically Separated into a Series of Blobpages

Instead of storing simple-large-object data in a blobspace, you can choose to store it in a dbspace. However, for a simple large object larger than two pages, performance improves when you store it in a blobspace. Simple large objects stored in a dbspace can share a page, but simple large objects stored in a blobspace do not share pages.

For information on how to determine the size of a blobpage, refer to "Determining Blobpage Size" on page 10-21.

Sbpages

An sbpage is the type of page that the database server uses to store smart large objects within a sbspace. For a description of sbspaces, refer to "Sbspaces" on page 9-12. Unlike blobpages, sbpages are not configurable. An sbpage is the same size as the database server page, which is usually 2 kilobytes on UNIX and 4 kilobytes on Windows.

The unit of allocation in an sbspace is an extent, whereas the unit of allocation in a blobspace is a blobpage. Just as with pages in a chunk, a certain number of smart large object extents compose a chunk in an sbspace, as Figure 9-5 illustrates. An extent is always entirely contained in a chunk and cannot cross chunk boundaries.

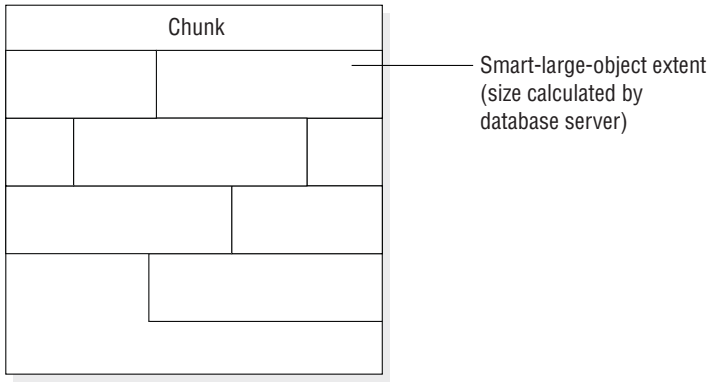


Figure 9-5. A Chunk in an Sbspace, Logically Separated into a Series of Extents

Smart large objects cannot be stored in a dbspace or blobspace. For more information, see "Sbspaces" on page 9-12, and sbspace structure in the disk structures and storage chapter of the *IBM Informix Dynamic Server Administrator's Reference*.

The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For more information, see "Extent Sizes for Sbspaces" on page 9-15.

Extents

When you create a table, the database server allocates a fixed amount of space to contain the data to be stored in that table. When this space fills, the database server must allocate space for additional storage. The physical unit of storage that the database server uses to allocate both the initial and subsequent storage space is called an *extent*.

Figure 9-6 illustrates the concept of an extent.

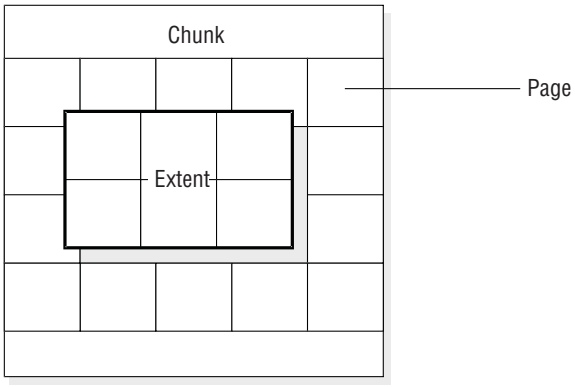


Figure 9-6. An Extent That Consists of Six Contiguous Pages on a Raw Disk Device

An extent consists of a collection of contiguous pages that store data for a given table. (See "Tables" on page 9-21.) Every permanent database table has two extent sizes associated with it. The *initial-extent* size is the number of kilobytes allocated to the table when it is first created. The *next-extent* size is the number of kilobytes allocated to the table when the initial extent (and any subsequent extents) becomes full. For permanent tables and user-defined temporary tables, the next-extent size begins to double after 16 extents have been added. For system-created temporary tables, the next-extent size begins to double after 4 extents have been added.

When you create a table, you can specify the size of the initial extent, as well as the size of the extents to be added as the table grows. You can also modify the size of an extent in a table in a dbspace, and you can modify the size of new subsequent extents. To specify the initial-extent size and next-extent size, use the CREATE TABLE and ALTER TABLE statements. For more information, see the *IBM Informix Guide to SQL: Syntax* and disk structures in the *IBM Informix Dynamic Server Administrator's Reference*.

When you create a table with a column for CLOB or BLOB data types, you also define extents for an sbspace. For more information, refer to "Storage Characteristics of Sbspaces" on page 9-14.

Figure 9-7 on page 9-8 shows how the database server allocates six pages for an extent:

- An extent is always entirely contained in a chunk; an extent cannot cross chunk boundaries.

- If the database server cannot find the contiguous disk space that is specified for the next-extent size, it searches the next chunk in the dbspace for contiguous space.

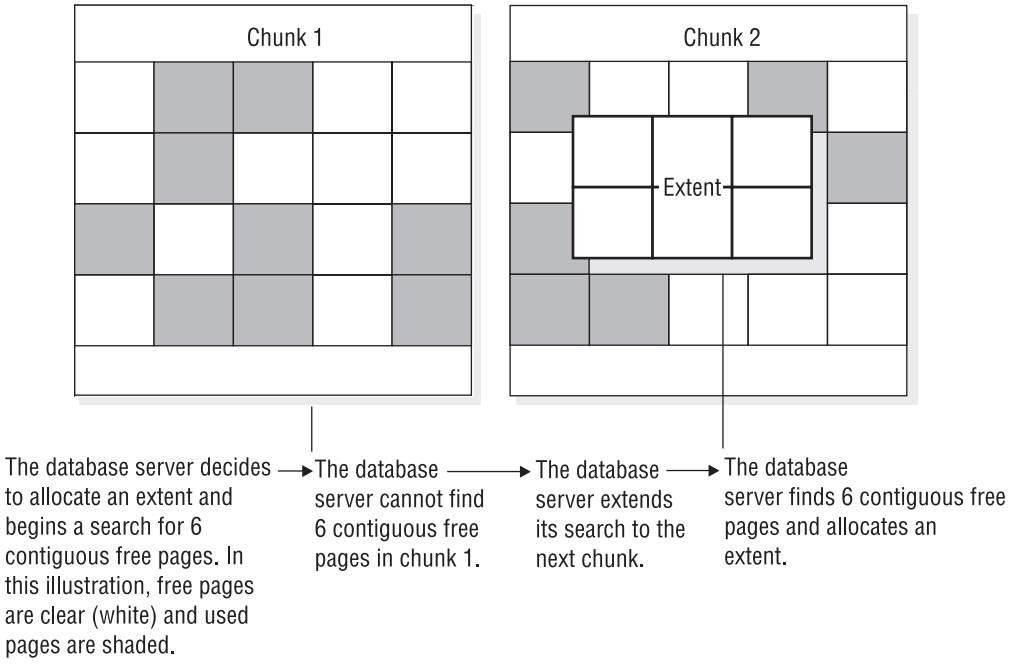


Figure 9-7. Process of Extent Allocation

Dbspaces

A *dbspace* is a logical unit that can contain between 1 and 32,766 chunks. Place databases, tables, logical-log files, and the physical log in dbspaces.

Control of Where Data Is Stored

A key responsibility of the database server administrator is to control where the database server stores data. By storing high-use access tables or *critical dbspaces* (root dbspace, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical data on separate physical devices, you ensure that when one of the disks holding noncritical data fails, the failure affects only the availability of data on that disk.

As Figure 9-8 on page 9-9 shows, to control the placement of databases or tables, you can use the IN *dbspace* option of the CREATE DATABASE or CREATE TABLE statements. (For more information, see “Tables” on page 9-21.)

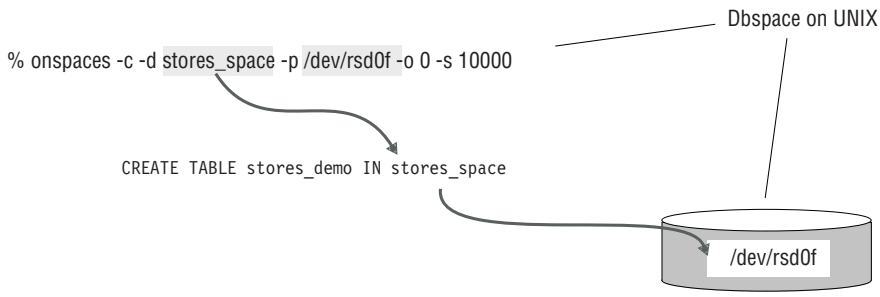


Figure 9-8. Controlling Table Placement with the CREATE TABLE... IN Statement

Before you create a database or table in a dbspace, you must first create the dbspace. For more information on how to create a dbspace, see "Creating a Dbspace that Uses the Default Page Size" on page 10-7.

A dbspace includes one or more chunks, as Figure 9-9 shows. You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor dbspace chunks for fullness and to anticipate the need to allocate more chunks to a dbspace. (See "Monitoring Disk Usage" on page 10-33.) When a dbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

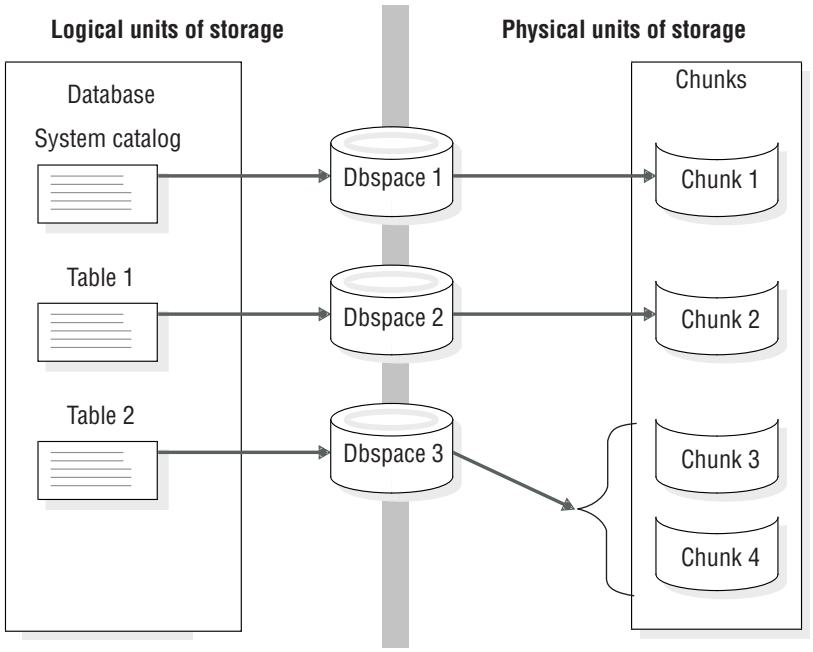


Figure 9-9. Dbspaces That Link Logical and Physical Units of Storage

The database server uses the dbspace to store databases and tables. (See "Tables" on page 9-21.)

When you create a standard or temporary dbspace, you can specify the page size for the dbspace. You cannot specify a page size for blobspaces, sbspaces, or external spaces. If you do not specify a page size, the size of the root dbspace is the default page size. For more information, see "Creating a Dbspace with a Non-Default Page Size" on page 10-10.

When you create a standard dbspace, you can specify the first and next extent sizes for the tblspace **tblspace** in the dbspace. Do this if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you need to place the tblspace **tblspace** extents in non-primary chunks. For more information, see "Specifying the First and Next Extent Sizes for the tblspace **tblspace**" on page 10-9.

You can mirror every chunk in a mirrored dbspace. As soon as the database server allocates a mirror chunk, it flags all space in that mirror chunk as full. See "Monitoring Disk Usage" on page 10-33.

For information on using ISA or **onspaces** to perform the following tasks, see Chapter 10, "Managing Disk Space," on page 10-1.

- Creating a dbspace
- Adding a chunk to a dbspace
- Renaming a dbspace
- Dropping a chunk
- Dropping a dbspace, blobspace, or sbspace

Root Dbspace

The root dbspace is the initial dbspace that the database server creates. The root dbspace is special because it contains reserved pages and internal tables that describe and track all physical and logical units of storage. (For more information on these topics, see "Tables" on page 9-21 and the disk structures and storage chapter in the *IBM Informix Dynamic Server Administrator's Reference*.) The initial chunk of the root dbspace and its mirror are the only chunks created during disk-space setup. You can add other chunks to the root dbspace after disk-space setup.

The following disk-configuration parameters in the ONCONFIG configuration file refer to the first (initial) chunk of the root dbspace:

- ROOTPATH
- ROOTOFFSET
- ROOTNAME
- MIRRORPATH
- MIRROROFFSET
- TBLTBLFIRST
- TBLTBLNEXT

The root dbspace is also the default dbspace location for any database created with the CREATE DATABASE statement.

The root dbspace is the default location for all temporary tables created by the database server to perform requested data management.

See "Size of the Root Dbspace" on page 9-31 for information on how much space to allocate for the root dbspace. You can also add extra chunks to the root dbspace after you set up database server disk space.

Temporary Dbspaces

A temporary dbspace is a dbspace reserved exclusively for the storage of temporary tables. You cannot mirror a temporary dbspace.

The database server never drops a temporary dbspace unless it is explicitly directed to do so. A temporary dbspace is temporary only in the sense that the database server does not preserve any of the dbspace contents when the database server shuts down abnormally.

Whenever you set up the database server, all temporary dbspaces are set up. The database server clears any tables that might remain since the last time that the database server shut down.

The database server does not perform logical or physical logging for temporary dbspaces. Because temporary dbspaces are not physically logged, fewer checkpoints and I/O operations occur, which improves performance.

The database server logs table creation, the allocation of extents, and the dropping of the table for a temporary table in a standard dbspace. In contrast, the database server does not log tables stored in temporary dbspaces. Logical-log suppression in temporary dbspaces reduces the number of log records to roll forward during logical recovery as well, thus improving the performance during critical down time.

Using temporary dbspaces to store temporary tables also reduces the size of your storage-space backup, because the database server does not back up temporary dbspaces.

If you have more than one temporary dbspace and execute a SELECT statement into a temporary table, the results of the query are inserted in round robin order.

For detailed instructions on how to create a temporary dbspace, see "Creating a Temporary Dbspace" on page 10-15.

Important: When the database server is running as a secondary database server, it requires a temporary dbspace to store any internal temporary tables generated by read-only queries.

Blobspaces

A blobspace is a logical storage unit composed of one or more chunks that store only TEXT and BYTE data. A blobspace stores TEXT and BYTE data in the most efficient way possible. You can store TEXT and BYTE columns associated with distinct tables (see "Tables" on page 9-21) in the same blobspace.

The database server writes data stored in a blobspace directly to disk. This data does not pass through resident shared memory. If it did, the volume of data could occupy so many of the buffer-pool pages that other data and index pages would be forced out. For the same reason, the database server does not write TEXT or BYTE objects that are assigned to a blobspace to either the logical or physical log. The database server logs blobspace objects by writing them directly from disk to the logical-log backup tapes when you back up the logical logs. Blobspace objects never pass through the logical-log files.

When you create a blobspace, you assign to it one or more chunks. You can add more chunks at any time. One of the tasks of a database server administrator is to monitor the chunks for fullness and anticipate the need to allocate more chunks to a blobspace. For instructions on how to monitor chunks for fullness, see "Monitoring Simple Large Objects in a Blobspace" on page 10-38. For instructions

on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see Chapter 10, "Managing Disk Space," on page 10-1.

For information about the structure of a blobspace, see the chapter on disk structures and storage in the *IBM Informix Dynamic Server Administrator's Reference*.

Sbspaces

An sbspace is a logical storage unit composed of one or more chunks that store *smart large objects*. Smart large objects consist of CLOB (character large object) and BLOB (binary large object) data types. User-defined data types can also use sbspaces. For more information about data types, refer to the *IBM Informix Guide to SQL: Reference*.

Advantages of Using Sbspaces

Sbspaces have the following advantages over blobspaces:

- They have read, write, and seek properties similar to a standard UNIX file.

  Programmers can use functions similar to UNIX and Windows functions to read, write, and seek smart large objects. Dynamic Server provides this smart-large-object interface in the DataBlade API and the Informix ESQL/C programming interface.
- They are recoverable.

  You can log all write operations on data stored in sbspaces. You can commit or rollback changes if a failure occurs during a transaction.
- They obey transaction isolation modes.

  You can lock smart large objects at different levels of granularity, and the lock durations obey the rules for transaction isolation levels. For more information on locking and concurrency, refer to your *IBM Informix Dynamic Server Performance Guide*.
- Smart large objects within table rows do not need to be retrieved in one statement.

  An application can store or retrieve smart large objects in pieces using either the DataBlade API or the Informix ESQL/C programming interface. For more information on the DataBlade API functions, refer to the *IBM Informix DataBlade API Function Reference*. For more information on the Informix ESQL/C functions, refer to the *IBM Informix ESQL/C Programmer's Manual*.

Sbspaces and Enterprise Replication

Before you define a replication server for Enterprise Replication, you must create an sbspace. Enterprise Replication spools the replicated data to smart large objects. Specify the sbspace name in the CDR_QDATA_SBSPACE configuration parameter. Enterprise Replication uses the default log mode with which the sbspace was created for spooling the row data. The CDR_QDATA_SBSPACE configuration parameter accepts multiple sbspaces, up to a maximum of 32 sbspaces. Enterprise Replication can support a combination of logging and non-logging sbspaces for storing spooled row data. For more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Metadata, User Data, and Reserved Area

As with blobspaces and dbspaces, when you create a sbspace, you assign to it one or more chunks. However, the first chunk of an sbspace always has three areas:

- *Metadata area*

Metadata identifies key aspects of the sbspace and each smart large object stored in the sbspace, and enables the database server to manipulate and recover smart large objects stored within.

- *User-data area*

  User data is the smart large object data stored in the sbspace by user applications. The chunk has up to two user-data areas.

- *Reserved area*

  The database server allocates space from the reserved area to either the metadata or user-data area when more space is needed. The chunk has up to two reserved areas.

For information on correctly allocating metadata and user data for sbspaces, see " Sizing Sbspace Metadata" on page 10-24 and the *IBM Informix Dynamic Server Performance Guide*.

When you add a chunk to an sbspace, you can specify whether it contains a metadata area and user-data area or whether to reserve the chunk exclusively for user data. You can add more chunks at any time. If you are updating smart large objects, I/O to the user data is much faster on raw disks than cooked chunk files. For instructions on how to create a sbspace, add chunks to a sbspace, or drop a chunk from a sbspace, see Chapter 10, "Managing Disk Space," on page 10-1.

Important: Sbspace metadata is always logged, regardless of the logging setting of the database.

Control of Where Data Is Stored

You specify the data type of a column when you create the table. For smart large objects, you specify CLOB, BLOB, or user-defined data types. As Figure 9-10 shows, to control the placement of smart large objects, you can use the IN *sbspace* option in the PUT clause of the CREATE TABLE statement.

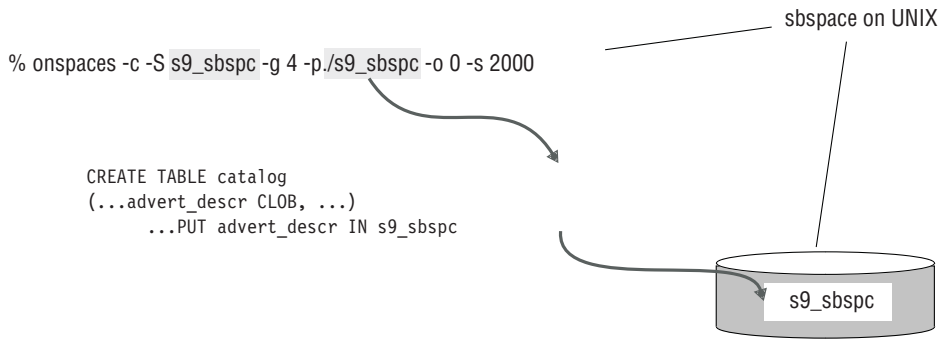


Figure 9-10. Controlling Smart-Large-Object Placement

Before you specify an sbspace in a PUT clause, you must first create the sbspace. For more information on how to create an sbspace with the **onspaces -c -S** command, see " Adding a Chunk to a Dbspace or Blobspace" on page 10-16. For more information on how to specify smart large object characteristics in the PUT clause, refer to the CREATE TABLE statement in the *IBM Informix Guide to SQL: Syntax*.

If you do not specify the PUT clause, the database server stores the smart large objects in the default sbspace that you specify in the SBSPACENAME configuration

parameter. For more information on SBSPACENAME, refer to the configuration parameter chapter of the *IBM Informix Dynamic Server Administrator's Reference*.

An sbspace includes one or more chunks, as Figure 9-11 shows. When an sbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor sbspace chunks for fullness and to anticipate the need to allocate more chunks to a sbspace. For more information on monitoring sbspaces, refer to your *IBM Informix Dynamic Server Performance Guide*.

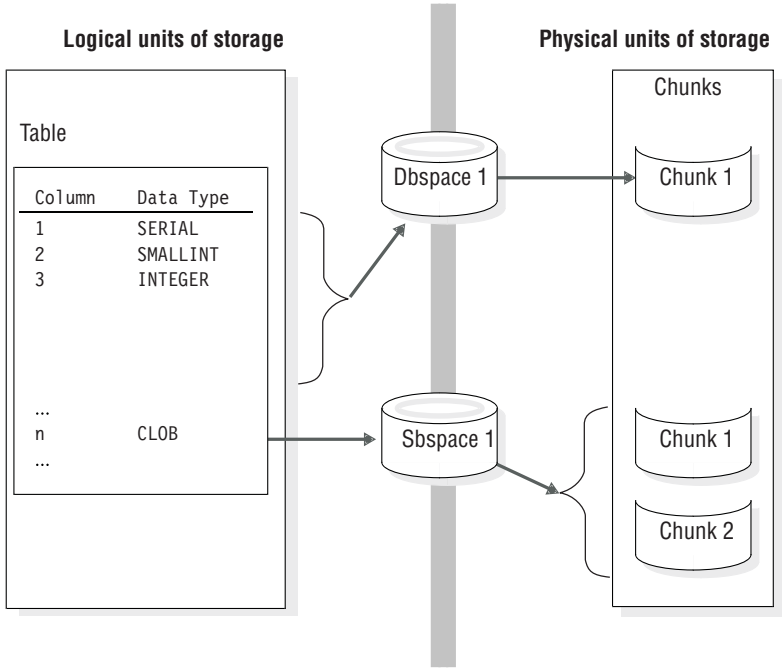


Figure 9-11. Sbspaces That Link Logical and Physical Units of Storage

The database server uses sbspaces to store table columns that contain smart large objects. The database server uses dbspaces to store the rest of the table columns.

You can mirror an sbspace to speed recovery in event of a media failure. For more information, refer to "Mirroring" on page 17-1.

For information on using **onspaces** to perform the following tasks, see Chapter 10, "Managing Disk Space," on page 10-1.

- Creating an sbspace
- Adding a chunk to an sbspace
- Altering storage characteristics of smart large objects
- Creating a temporary sbspace
- Dropping an sbspace

Storage Characteristics of Sbspaces

As the database server administrator, you can use the system default values for these storage characteristics, or you can specify them in the **-Df** tags when you create the sbspace with **onspaces -c.** Later on, you can change these sbspace

characteristics with the **onspaces -ch** option. The administrator or programmer can override these default values for storage characteristics and attributes for individual tables.

Extent Sizes for Sbspaces

Similar to extents in a table, an extent in an sbspace consists of a collection of contiguous pages that store smart large object data.

The unit of allocation in an sbspace is an extent. The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For example, if an operation asks to write 30 kilobytes, the database server tries to allocate an extent the size of 30 kilobytes.

Important: For most applications, you should use the values that the database server calculates for the extent size.

If you know the size of the smart large object, you can use one of the following functions to set the extent size. The database server allocates the entire smart large object as one extent (if an extent of that size is available in the chunk):

- The DataBlade API **mi_lo_specset_estbytes()** function

  For more information on the DataBlade API functions for smart large objects, refer to the *IBM Informix DataBlade API Function Reference*.
- The Informix ESQL/C **ifx_lo_specset_estbytes** function

  For more information on the Informix ESQL/C functions for smart large objects, refer to the *IBM Informix ESQL/C Programmer's Manual*.

For information about tuning extent sizes, see smart large objects in the chapter on configuration effects on I/O utilization in your *IBM Informix Dynamic Server Performance Guide*.

Average Smart-Large-Object Size

Smart large objects usually vary in length. You can provide an average size of your smart large objects to calculate space for an sbspace. You specify this average size with the AVG_LO_SIZE tag of the **onspaces -c -Df** option.

To specify the size and location of the metadata area, specify the **-Ms** and **-Mo** flags in the **onspaces** command. If you do not use the **-Ms** flag, the database server uses the value of AVG_LO_SIZE to estimate the amount of space to allocate for the metadata area. For more information, refer to " Sizing Sbspace Metadata" on page 10-24.

Buffering Mode

When you create an sbspace, the default buffering mode is on, which means to use the buffer pool in the resident portion of shared memory.

As the database administrator, you can specify the buffering mode with the BUFFERING tag of the **onspaces -c -Df** option. The default is "buffering=ON", which means to use the buffer pool. If you turn off buffering, the database server uses private buffers in the virtual portion of shared memory.

Important: In general, if read and write operations to the smart large objects are less than 8 kilobytes, do not specify a buffering mode when you create the sbspace. If you are reading or writing short blocks of data, such as 2 kilobytes or 4 kilobytes, leave the default of "buffering=ON" to obtain better performance.

For information about when to use private buffers, see the section on light-weight I/O operations in the chapter on configuration effects on I/O utilization in your *IBM Informix Dynamic Server Performance Guide*.

Last-Access Time

When you create an sbspace, you can specify whether or not the database server should keep the last time that the smart large object was read or updated with the ACCESSTIME tag of the **onspaces -c -Df** option. The default is "ACCESSTIME=OFF". The database server keeps this last-access time in the metadata area.

For more information on how programmers use this last-access time, refer to the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Lock Mode

When you create an sbspace, you can specify whether or not the database server should lock the whole smart large object or a range of bytes within a smart large object with the LOCK_MODE tag of the **onspaces -c -Df** option. The default is "LOCK_MODE=BLOB", which means to lock the entire smart large object. For more information, refer to the locking chapter in your *IBM Informix Dynamic Server Performance Guide*.

Logging

When you create an sbspace, you can specify whether or not to turn on logging for the smart large objects. The default is no logging. For more information, refer to "Logging Sbspaces and Smart Large Objects" on page 13-7.

Important: When you use logging databases, turn logging on for the sbspaces. If a failure occurs that requires log recovery, you can keep the smart large objects consistent with the rest of the database.

You specify the logging status with the LOGGING tag of the **onspaces -c -Df** option. The default is "LOGGING=off". You can change the logging status with the **onspaces -ch -Df** option. You can override this logging status with the PUT clause in the SQL statements CREATE TABLE or ALTER TABLE. For more information about these SQL statements, refer to the *IBM Informix Guide to SQL: Syntax*.

The programmer can override this logging status with functions that the DataBlade API and Informix ESQL/C provide. For more information on the DataBlade API functions for smart large objects, refer to the *IBM Informix DataBlade API Function Reference*. For more information on the Informix ESQL/C functions for smart large objects, refer to the *IBM Informix ESQL/C Programmer's Manual*.

When you turn on logging for an sbspace, the smart large objects pass through the resident portion of shared memory. Although applications can retrieve pieces of a smart large object, you still need to consider the larger size of data that might pass through the buffer pool and logical-log buffers. For more information, refer to "Accessing Smart Large Objects" on page 7-31.

Levels of Inheritance for Sbspace Characteristics

The four levels of inheritance for sbspace characteristics are system, sbspace, column, and smart large objects. You can use the system default values for sbspace attributes, or override them for specific sbspaces, columns in a table, or smart large objects. Figure 9-12 on page 9-17 shows the storage-characteristics hierarchy for a

smart large object.

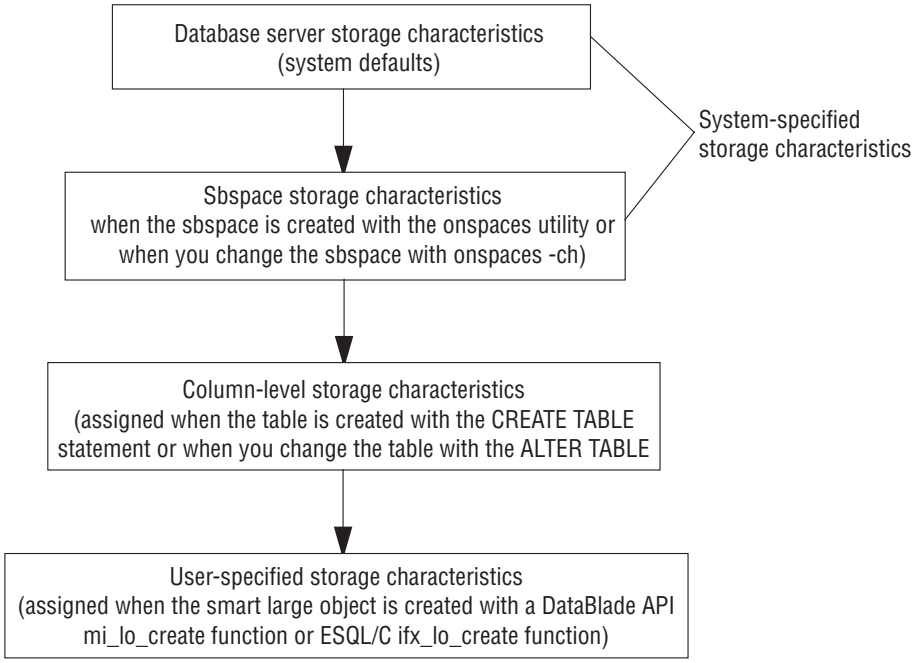


Figure 9-12. Storage-Characteristics Hierarchy

Figure 9-12 shows that you can override the system default in the following ways:

- Use the **-Df** tags of the **onspaces -c -S** command to override the system default for a specific sbspace.

  You can later change these sbspace attributes for the sbspace with the **onspaces -ch** option. For more information on valid ranges for the **-Df** tags, refer to the **onspaces** section in the *IBM Informix Dynamic Server Administrator's Reference*.
- You override the system default for a specific column when you specify these attributes in the PUT clause of the CREATE TABLE or ALTER TABLE statements**.**

  For more information on these SQL statements, refer to the *IBM Informix Guide to SQL: Syntax*.
- The programmer can override the default values for sbspace attributes for specific smart large objects with functions that the DataBlade API and Informix ESQL/C programming interface provide.

More Information About Sbspaces

Table 9-1 lists sources of information about various tasks related to using and managing sbspaces.

Table 9-1. Finding Information for Sbspace Tasks

Task	Reference
Setting memory configuration parameters for smart large objects	Chapter 8, "Managing Shared Memory," on page 8-1
Understanding sbpages	"Sbpages" on page 9-6
Specifying I/O characteristics for an sbspace	**onspaces** option in "Storage Characteristics of Sbspaces" on page 9-14
Allocating space for an sbspace	" Creating an Sbspace" on page 10-23

Table 9-1. Finding Information for Sbspace Tasks (continued)

Task	Reference
Adding a chunk to an sbspace	"Adding a Chunk to an Sbspace" on page 10-24
Defining or altering storage characteristics for a smart large object	"Altering Storage Characteristics of Smart Large Objects" on page 10-25 PUT clause of CREATE TABLE or ALTER TABLE statement in *IBM Informix Guide to SQL: Syntax*
Monitoring sbspaces	"Monitoring Sbspaces" on page 10-41 Chapter on table performance considerations in *IBM Informix Dynamic Server Performance Guide*
Setting up logging for an sbspace	"Logging Sbspaces and Smart Large Objects" on page 13-7
Backing up an sbspace	"Backing Up Sbspaces" on page 14-4
Checking consistency of an sbspace	"Validating Metadata" on page 24-3
Understanding an sbspace structure	Chapter on disk structures in the *IBM Informix Administrator's Reference*
Using **onspaces** for sbspaces	Chapter on utilities in the *IBM Informix Administrator's Reference*
Creating a table with CLOB or BLOB data types	*IBM Informix Guide to SQL: Syntax*
Accessing smart large objects in an application	*IBM Informix DataBlade API Programmer's GuideIBM Informix ESQL/C Programmer's Manual*
Calculating the metadata area size Improving metadata I/O Changing storage characteristics	Chapter on table performance in *IBM Informix Dynamic Server Performance Guide*
Understanding smart-large-object locking	Chapter on locking in *IBM Informix Dynamic Server Performance Guide*
Configuring sbspaces for temporary smart large objects	Chapter on configuration effects on I/O activity in *IBM Informix Dynamic Server Performance Guide*

Temporary Sbspaces

Use a *temporary sbspace* to store temporary smart large objects without metadata logging and user-data logging. If you store temporary smart large objects in a standard sbspace, the metadata is logged. Temporary sbspaces are similar to temporary dbspaces. To create a temporary sbspace, use the **onspaces -c -S** command with the **-t** option. For more information, see "Creating a Temporary Sbspace" on page 10-25.

You can store temporary large objects in a standard sbspace or temporary sbspace.

- If you specify a temporary sbspace in the SBSPACETEMP parameter, you can store temporary smart large objects there.
- If you specify a standard sbspace in the SBSPACENAME parameter, you can store temporary and permanent smart large objects there.
- If you specify a temporary sbspace name in the CREATE TEMP TABLE statement, you can store temporary smart large objects there.
- If you specify a permanent sbspace name in the CREATE TABLE statement, you can store temporary smart large objects there.
- If you omit the SBSPACETEMP and SBSPACENAME parameters and create a smart large object, error message `-12053` might display.

- If you specify a temporary sbspace in the SBSPACENAME parameter, you cannot store a *permanent* smart large object in that sbspace. You can store temporary smart large objects in that sbspace.

Comparison of Temporary and Standard Sbspaces

Table 9-2 compares standard and temporary sbspaces.

Table 9-2. Temporary and Standard Sbspaces

Characteristics	Standard Sbspace	Temporary Sbspace
Stores smart large objects	Yes	No
Stores temporary smart large objects	Yes	Yes
Logs metadata	Metadata is always logged	Metadata is not logged
Logs user data	User data is not logged for temporary smart large objects but is logged for permanent smart large objects if LOGGING=ON	User data is not logged Creation and deletion of space, and addition of chunks is logged
Fast recovery	Yes	No (the sbspace is emptied when the database server restarts) To set up shared memory without cleaning up temporary smart large objects, specify **oninit -p**. If you keep the temporary large objects, their state is indeterminate.
Backup and restore	Yes	No
Add and drop chunks	Yes	Yes
Configuration parameter	SBSPACENAME	SBSPACETEMP

Temporary Smart Large Objects

Use *temporary smart large objects* to store text or image data (CLOB or BLOB) that do not require restoring from a backup or log replay in fast recovery. Temporary smart large objects last for the user session and are much faster to update than smart large objects.

You create a temporary smart large object in the same way as a permanent smart large object, except you set the LO_CREATE_TEMP flag in the **ifx_lo_specset_flags** or **mi_lo_specset_flags** function. Use **mi_lo_copy** or **ifx_lo_copy** to create a permanent smart large object from a temporary smart large object. For details on creating temporary smart large objects, see the *IBM Informix DataBlade API Programmer's Guide*.

Important: Store pointers to temporary large objects in temporary tables only. If you store them in standard tables and reboot the database server, it results in an error that says that the large object does not exist.

Table 9-3 compares standard and temporary smart large objects.

Table 9-3. Temporary and Standard Smart Large Objects

Characteristics	Smart Large Object	Temporary Smart Large Object
Creation flags	LO_CREATE_LOG or LO_CREATE_NOLOG	LO_CREATE_TEMP
Rollback	Yes	No

Table 9-3. Temporary and Standard Smart Large Objects (continued)

Characteristics	Smart Large Object	Temporary Smart Large Object
Logging	Yes, if turned on	No
Duration	Permanent (until user deletes it)	Deleted at end of user session or transaction
Table type stored in	Permanent or temporary table	Temporary tables

Extspaces

An extspace is a logical name associated with an arbitrary string that signifies the location of external data. The resource that the extspace references depends on a user-defined access method for accessing its contents.

For example, a database user might require access to binary files encoded in a proprietary format. First, a developer creates an *access method*, which is a set of routines that access the data. These routines are responsible for all interaction between the database server and the external file. A DBA then adds an extspace that has the file as its target to the database. After the DBA creates a table in the extspace, the users can access the data in the proprietary files via SQL statements. To locate those files, use the extspace information.

An extspace need not be a filename. For example, it can be a network location. The routines that access the data can use information found in the string associated with the extspace in any manner.

For more information on user-defined access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*. For more information on creating functions and primary access methods, see the *IBM Informix Guide to SQL: Syntax*.

Databases

A database is a logical storage unit that contains tables and indexes. (See "Tables" on page 9-21.) Each database also contains a system catalog that tracks information about many of the elements in the database, including tables, indexes, SPL routines, and integrity constraints.

A database resides in the dbspace specified by the CREATE DATABASE statement. When you do not explicitly name a dbspace in the CREATE DATABASE statement, the database resides in the root dbspace. When you *do* specify a dbspace in the CREATE DATABASE statement, this dbspace is the location for the following tables:

- Database system catalog tables
- Any table that belongs to the database

Figure 9-13 on page 9-21 shows the tables contained in the stores_demo database.

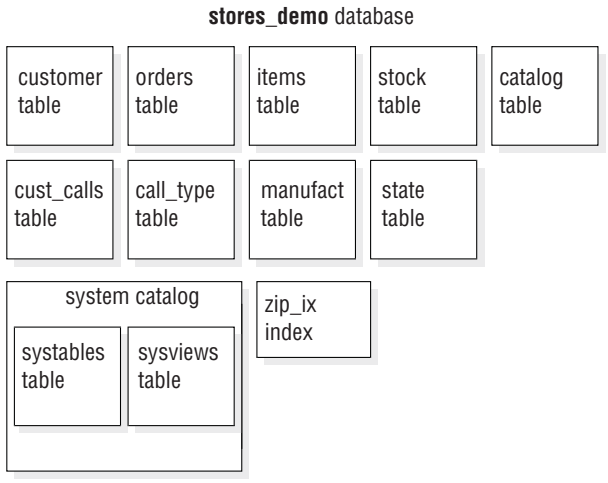


Figure 9-13. The stores_demo Database

The size limits that apply to databases are related to their location in a dbspace. To be certain that all tables in a database are created on a specific physical device, assign only one chunk to the device, and create a dbspace that contains only that chunk. Place your database in that dbspace. When you place a database in a chunk assigned to a specific physical device, the database size is limited to the size of that chunk.

For instructions on how to list the databases that you create, see "Displaying Databases" on page 10-33.

Tables

In relational database systems, a table is a row of column headings together with zero or more rows of data values. The row of column headings identifies one or more columns and a data type for each column.

When you create a table, the database server allocates disk space for the table in a block of pages called an extent. (See "Extents" on page 9-7.) You can specify the size of both the first and any subsequent extents.

You can place the table in a specific dbspace by naming the dbspace when they create the table (usually with the IN *dbspace* option of CREATE TABLE). When you do not specify the dbspace, the database server places the table in the dbspace where the database resides.

You can also:

- Fragment a table over more than one dbspace. However, you cannot put fragments into different page-size dbspaces. All fragments might need to be the same page size.

  You must define a distribution scheme for the table that specifies which table rows are located in which dbspaces.
- Create multiple partitions of a fragmented table within a single dbspace if the fragmented tables uses an expression-based or round-robin distribution scheme.

A table or table fragment resides completely in the dbspace in which it was created. The database server administrator can use this fact to limit the growth of a table by placing a table in a dbspace and then refusing to add a chunk to the dbspace when it becomes full.

For more information about distribution schemes, see the *IBM Informix Database Design and Implementation Guide*. For information on how to fragment tables and indexes over multiple disks to improve performance, concurrency, data availability, and backups, refer to your *IBM Informix Dynamic Server Performance Guide*.

A table, composed of extents, can span multiple chunks, as Figure 9-14 shows.

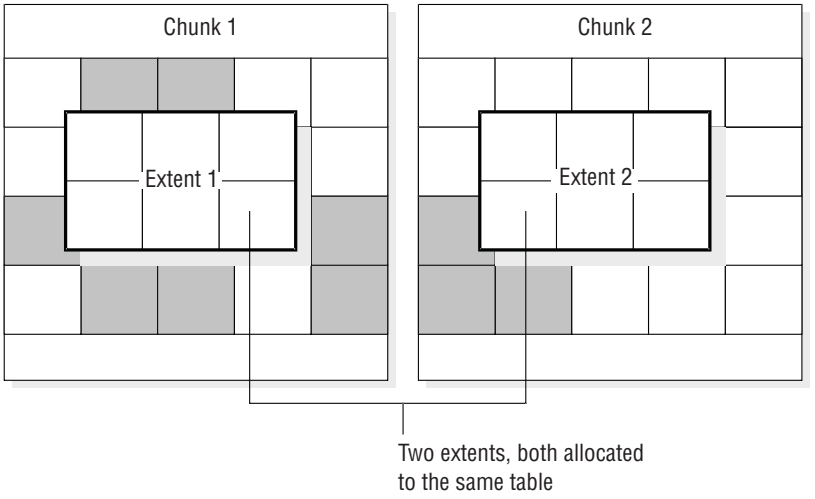


Figure 9-14. Table That Spans More Than One Chunk

Simple large objects reside in blobpages in either the dbspace with the data pages of the table or in a separate blobspace. When you use the Optical Subsystem, you can also store simple large objects in an optical storage subsystem.

For advice on where to store your tables, see "Disk-Layout Guidelines" on page 9-33 and your *IBM Informix Dynamic Server Performance Guide*.

Damaged Tables

The following items can damage a table:

- An incorrect buffer flush
- A user error
- Mounting a files system or another chunk on top of a chunk
- Deleting or updating when the scope of the change is not as narrow as you need

Damaged indexes can cause a table to appear damaged, even though the table is okay.

oncheck commands cannot fix most damaged tables. If a page is damaged, **oncheck** can detect and try to fix the page, but cannot correct the data within the page.

Table Types for Dynamic Server

You can create logging or nonlogging tables in a logging database on Dynamic Server. The two table types are STANDARD (logging tables) and RAW (nonlogging tables). The default standard table is like a table created in earlier versions without a special keyword specified. You can create either a STANDARD or RAW table and change tables from one type to another.

In a nonlogging database, both STANDARD tables and RAW tables are nonlogging. In a nonlogging database, the only difference between STANDARD and RAW tables is that RAW tables do not support primary constraints, unique constraints, and rollback. However, these tables can be indexed and updated.

Table 9-4 lists the properties of the types of tables available with Dynamic Server. The flag values are the hexadecimal values for each table type in the **flags** column of **systables**.

Table 9-4. Table Types for Dynamic Server

Characteristic	STANDARD	RAW	TEMP
Permanent	Yes	Yes	No
Logged	Yes	No	Yes
Indexes	Yes	Yes	Yes
Rollback	Yes	No	Yes
Recoverable	Yes	Yes, if not updated	No
Restorable	Yes	Yes, if not updated	No
Loadable	Yes	Yes	Yes
Enterprise Replication	Yes	No	No
Flag Value	None	0x10	None

Standard Permanent Tables

A STANDARD table is the same as a table in a logged database that the database server creates. STANDARD tables do not use light appends. All operations are logged, record by record, so STANDARD tables can be recovered and rolled back. You can back up and restore STANDARD tables. Logging enables updates since the last physical backup to be applied when you perform a warm restore or point-in-time restore. Enterprise Replication is allowed on STANDARD tables.

A STANDARD table is the default type on both logging and nonlogging databases. STANDARD tables are logged if stored in a logging database but are not logged if stored in a nonlogging database.

RAW Tables

RAW tables are nonlogging permanent tables that are similar to tables in a nonlogging database. Update, insert, and delete operations on rows in a RAW table are supported but are not logged. You can define indexes on RAW tables, but they do not support primary key constraints or unique constraints. Light appends are not supported for loading RAW tables, except in High Performance Loader operations and in queries that specify `INTO TEMP ... WITH NO LOG.`

A RAW table has the same attributes, whether it is stored in a logging database or in a nonlogging database. If you update a RAW table, you cannot reliably restore

the data unless you perform a level-0 backup after the update. If the table has not been updated since that backup, you can restore the RAW table from the last physical backup, but backing up only the logical logs is not sufficient for a RAW table to be recoverable. Fast recovery can roll back incomplete transactions on STANDARD tables but not on RAW tables. For information on creating and altering RAW tables, see the *IBM Informix Guide to SQL: Syntax*.

RAW tables are intended for the initial loading and validation of data. To load RAW tables, you can use any loading utility, including **dbexport** or the High-Performance Loader (HPL) in express mode. If an error or failure occurs while loading a RAW table, the resulting data is whatever was on the disk at the time of the failure.

Recommendation: Do not use RAW tables within a transaction. After you have loaded the data, use the ALTER TABLE statement to change the table to type STANDARD and perform a level-0 backup before you use the table in a transaction.

Warning: Do not use Enterprise Replication on *RAW* or *TEMP* tables.

There are some restrictions when using RAW tables in a high-availability cluster environment. Because modifications made to RAW tables are not logged, and because secondary servers (including HDR, RSS and SDS) use log records to stay synchronized with the primary server, you are restricted from performing certain operations on RAW tables:

- On a primary server, RAW tables can be created, dropped, and accessed; however, altering the table mode from unlogged to logged, or from logged to unlogged, is not allowed. Altering a table's mode in a high-availability cluster environment yields error -19845.
- On secondary servers (HDR, SDS, or RSS), RAW tables are not accessible for any operation. Attempting to access a RAW table from SQL yields error -19846.

Temp Tables

Temp tables are temporary, logged tables that are dropped when the user session closes, the database server shuts down, or on reboot after a failure. Temp tables support indexes, constraints, and rollback. You cannot recover, back up, or restore temp tables. Temp tables support bulk operations such as light appends, which add rows quickly to the end of each table fragment. For more information about light appends, refer to your *IBM Informix Dynamic Server Performance Guide*.

For more information, see "Temporary Tables" on page 9-25.

Properties of Table Types

This section discusses loading tables, fast recovery, and backup and restore of table types.

Loading of Data Into a Table

Dynamic Server creates STANDARD tables that use logging by default. Data warehousing applications can have huge tables that take a long time to load. Nonlogging tables are faster to load than logging tables. You can use the CREATE RAW TABLE statement to create a RAW table or use the ALTER TABLE statement to change a STANDARD table to RAW before loading the table. After you load the table, run UPDATE STATISTICS on it.

For more information about how to improve the performance of loading very large tables, see your *IBM Informix Dynamic Server Performance Guide*. For more information on using ALTER TABLE to change a table from logging to nonlogging, see the *IBM Informix Guide to SQL: Syntax*.

Fast Recovery of Table Types

Table 9-5 shows fast recovery scenarios for the table types available with Dynamic Server.

Table 9-5. Fast Recovery of Table Types

Table Type	Fast Recovery Behavior
Standard	Fast recovery is successful. All committed log records are rolled forward, and all incomplete transactions are rolled back.
RAW	If a checkpoint completed since the raw table was modified last, all the data is recoverable. Inserts, updates, and deletions that occurred after the last checkpoint are lost. Incomplete transactions in a RAW table are not rolled back.

Backup and Restore of RAW Tables

Table 9-6 discusses backup scenarios for the table types available on Dynamic Server.

Table 9-6. Backing Up Tables on Dynamic Server

Table Type	Backup Allowed?
Standard	Yes.
Temp	No.
RAW	Yes. If you update a RAW table, you must back it up so that you can restore all the data in it. Backing up only the logical logs is not enough.

Important: After you load a RAW table or change a RAW table to type STANDARD, you must perform a level-0 backup.

Table 9-7 shows restore scenarios for these table types.

Table 9-7. Restoring Tables on Dynamic Server

Table Type	Restore Allowed?
Standard	Yes. Warm restore, cold restore, and point-in-time restore work.
Temp	No.
RAW	When you restore a RAW table, it contains only data that was on disk at the time of the last backup. Because RAW tables are not logged, any changes that occurred since the last backup are not restored.

Temporary Tables

The database server needs to provide disk space for temporary tables of the following two kinds:

- Temporary tables that you create with an SQL statement, such as CREATE TEMP TABLE... or SELECT .... INTO SCRATCH

- Temporary tables that the database server creates as it processes a query

Make sure that your database server has configured enough temporary space for both user-created and database server-created temporary tables. Some uses of the database server might require as much temporary storage space as permanent storage space, or more.

By default, the database server stores temporary tables in the root dbspace. If you decide not to store your temporary tables in the root dbspace, use the **DBSPACETEMP** environment variable or the DBSPACETEMP configuration parameter to specify a list of dbspaces for temporary tables.

Temporary Tables That You Create

You can create temporary tables with any of the following SQL statements:

- TEMP TABLE option of the CREATE TABLE statement
- INTO TEMP clause of the SELECT statement, such as `SELECT * FROM customer INTO TEMP` *cust_temp*

Only the session that creates a temporary table can use the table. When the session exits, the table is dropped automatically.

When you create a temporary table, the database server uses the following criteria:

- If the query used to populate the TEMP table produces no rows, the database server creates an empty, unfragmented table.
- If the rows that the query produces do not exceed 8 kilobytes, the temporary table resides in only one dbspace.
- If the rows exceed 8 kilobytes, the database server creates multiple fragments and uses a round-robin fragmentation scheme to populate them unless you specify a fragmentation method and location for the table.

If you use the CREATE TEMP and SELECT...INTO TEMP SQL statements and DBSPACETEMP has been set:

- LOGGING dbspaces in the list are used to create the tables that specify or imply the WITH LOG clause.
- NON-LOGGING temporary dbspaces in the list are used to create the tables that specify the WITH NO LOG clause.

When CREATE TEMP and SELECT...INTO TEMP SQL statements are used and DBSPACETEMP has not been set or does not contain the correct type of dbspace, Dynamic Server uses the dbspace of the database to store the temporary table. See the *IBM Informix Guide to SQL: Syntax* for more information.

Where User-Created Temporary Tables are Stored: If your application lets you specify the location of a temporary table, you can specify either logging spaces or nonlogging spaces that you create exclusively for temporary tables. SCRATCH tables are not logged and must be created in temporary spaces.

For information about creating temporary dbspaces and dbslices, refer to the **onspaces** section in the *IBM Informix Administrator's Reference*.

If you do not specify the location of a temporary table, the database server stores the temporary table in one of the spaces that you specify as an argument to the DBSPACETEMP configuration parameter or environment variable. The database server keeps track of the name of the last dbspace that it used for a temporary

table. When the database server receives another request for temporary storage space, it uses the next available dbspace to spread I/O evenly across the temporary storage space.

For information about where the database stores temporary tables when you do not list any spaces as an argument to DBSPACETEMP, see the DBSPACETEMP section in the *IBM Informix Administrator's Reference*.

When you use an application to create a temporary table, you can use the temporary table until the application exits or performs one of the following actions:

- Closes the database in which the table was created and opens a database in a different database server
- Closes the database in which the table was created
- Explicitly drops the temporary table

Temporary Tables That the Database Server Creates

The database server sometimes creates temporary tables while running queries against the database or backing it up. The database server might create a temporary table in any of the following circumstances:

- Statements that include a GROUP BY or ORDER BY clause
- Statements that use aggregate functions with the UNIQUE or DISTINCT keywords
- SELECT statements that use auto-index or hash joins
- Complex CREATE VIEW statements
- DECLARE statements that create a scroll cursor
- Statements that contain correlated subqueries
- Statements that contain subqueries that occur within an IN or ANY clause
- CREATE INDEX statements

When the process that initiated the creation of the table is complete, the database server deletes the temporary tables that it creates.

If the database server shuts down without removing temporary tables, it performs temporary table cleanup the next time shared-memory is set up. To initialize shared memory without temporary table cleanup, execute **oninit** with the **-p** option.

Important: In addition to temporary tables, the database server uses temporary disk space to store the before images of data that are overwritten while backups are occurring and overflow from query processing that occurs in memory. Make sure that you have configured enough temporary space for all uses.

Where Database Server-Created Temporary Tables are Stored: When the database server creates a temporary table, it stores the temporary table in one of the dbspaces that you specify in the DBSPACETEMP configuration parameter or the DBSPACETEMP environment variable. The environment variable supersedes the configuration parameter.

When you do not specify any temporary dbspaces in DBSPACETEMP, or the temporary dbspaces that you specify have insufficient space, the database server creates the table in a standard dbspace according to the following rules:

- If you created the temporary table with CREATE TEMP TABLE, the database server stores this table in the dbspace that contains the database to which the table belongs.
- If you created the temporary table with the INTO TEMP option of the SELECT statement, the database server stores this table in the root dbspace.

For more information, see "Creating a Temporary Dbspace" on page 10-15.

Tblspaces

Database server administrators sometimes need to track disk use by a particular table. A *tblspace* contains all the disk space allocated to a given table or table fragment (if the table is fragmented). A separate tblspace contains the disk space allocated for the associated index.

A tblspace, for example, does not correspond to any particular part of a chunk or even to any particular chunk. The indexes and data that make up a tblspace might be scattered throughout your chunks. The tblspace, however, represents a convenient accounting entity for space across chunks devoted to a particular table. (See "Tables" on page 9-21.)

Maximum Number of Tblspaces in a Table

You can specify a maximum of 2**20 (or 1,048,576) tblspaces in a table.

Table and Index Tblspaces

The table tblspace contains the following types of pages:

- Pages allocated to data
- Pages allocated to indexes
- Pages used to store TEXT or BYTE data in the dbspace (but not pages used to store TEXT or BYTE data in a blobspace)
- Bitmap pages that track page use within the table extents

The index tblspace contains the following types of pages:

- Pages allocated to indexes
- Bitmap pages that track page use within the index extents

Figure 9-15 on page 9-29 illustrates the tblspaces for three tables that form part of the **stores_demo** database. Only one table (or table fragment) exists per tblspace. An index resides in a separate tblspace from the associated table. Blobpages represent TEXT or BYTE data stored in a dbspace.

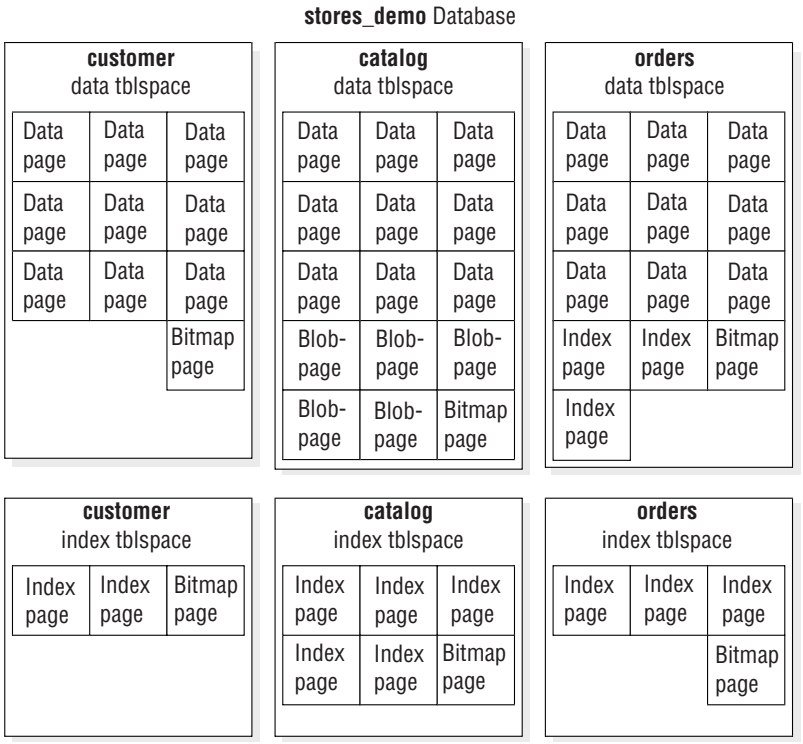


Figure 9-15. Sample Tblspaces in the stores_demo Database

Extent Interleaving

The database server allocates the pages that belong to a tblspace as extents. Although the pages within an extent are contiguous, extents might be scattered throughout the dbspace where the table resides (even on different chunks).

Figure 9-16 on page 9-30 depicts this situation with two noncontiguous extents that belong to the tblspace for **table_1** and a third extent that belongs to the tblspace for **table_2**. A **table_2** extent is positioned between the first table_1 extent and the second **table_1** extent. When this situation occurs, the extents are interleaved. Because sequential access searches across **table_1** require the disk head to seek across the **table_2** extent, performance is worse than if the **table_1** extents were contiguous. For instructions on how to avoid and eliminate interleaving extents, see your *IBM Informix Dynamic Server Performance Guide*.

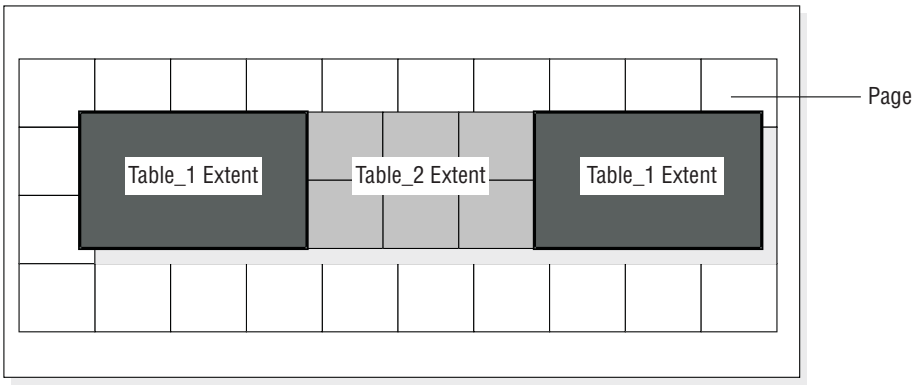


Figure 9-16. Three Extents That Belong to Two Different Tblspaces in a Single Dbspace

Table Fragmentation and Data Storage

The fragmentation feature gives you additional control over where the database stores data. You are not limited to specifying the locations of individual tables and indexes. You can also specify the location of table and index *fragments*, which are different parts of a table or index that reside on different storage spaces. You can fragment the following storage spaces:

- Dbspaces
- Sbspaces

Usually you fragment a table when you initially create it. The CREATE TABLE statement takes one of the following forms:

```
CREATE TABLE tablename ... FRAGMENT BY ROUND ROBIN IN dbspace1,
 dbspace2, dbspace3;

CREATE TABLE tablename ...FRAGMENT BY EXPRESSION
   <Expression 1> in dbspace1,
   <Expression 2> in dbspace2,
   <Expression 3> in dbspace3;
```

The FRAGMENT BY ROUND ROBIN and FRAGMENT BY EXPRESSION keywords refer to two different distribution schemes. Both statements associate fragments with dbspaces.

When you fragment a table, you can also create multiple partitions of the table within the same dbspace, as shown in this example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
         PARTITION part1 (a >=0 AND a < 5) in dbs1,
         PARTITION part2 (a >=5 AND a < 10) in dbs1
          ...
       ;
```

Figure 9-17 on page 9-31 illustrates the role of fragments in specifying the location of data.

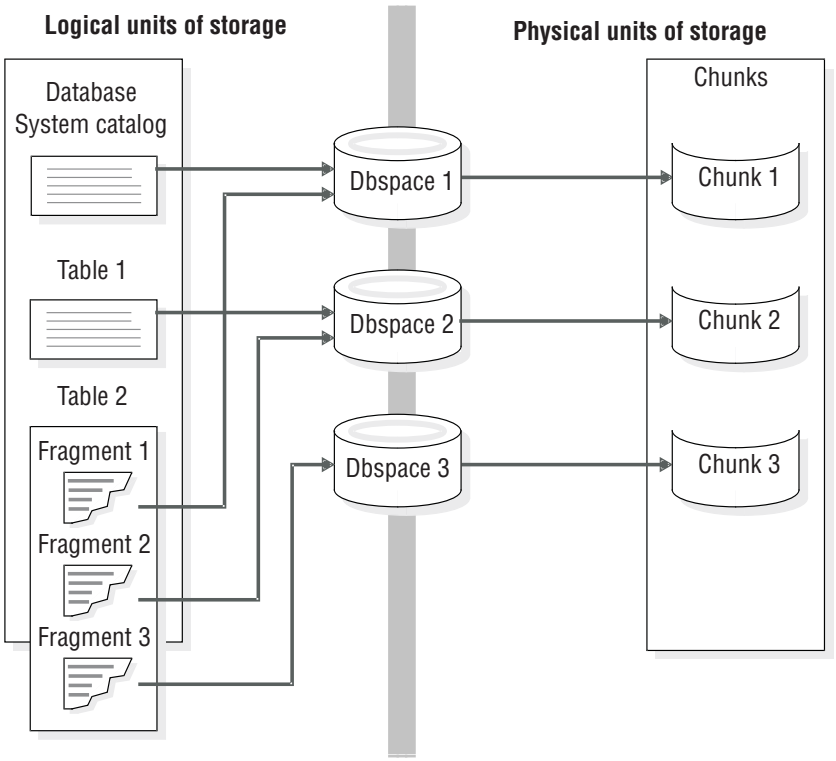


Figure 9-17. Dbspaces That Link Logical Units (Including Table Fragments) and Physical Units of Storage

For information on spaces and partitions, see Chapter 10, "Managing Disk Space," on page 10-1.

For more information about fragmentation, refer to the *IBM Informix Database Design and Implementation Guide* and the *IBM Informix Dynamic Server Performance Guide*.

Amount of Disk Space Needed to Store Data

To determine how much disk space you need, follow these steps:

To determine how much disk space you need

1. Calculate the size requirements of the root dbspace.
2. Estimate the total amount of disk space to allocate to all the database server databases, including space for overhead and growth.

The following sections explain these steps.

Size of the Root Dbspace

To calculate the size of the root dbspace, take the following storage structures into account:

- The physical log (200 kilobytes minimum)
- The logical-log files (200 kilobytes minimum)
- Temporary tables
- Data

- System databases (**sysmaster**, **sysutils**, **syscdr**, **sysuuid**) and system catalogs (the size varies between versions)
- Reserved pages (~24 kilobytes)
- Tblspace tblspace (100 to 200 kilobytes minimum)
- Extra space

This estimate is the root dbspace size before you initialize the database server. The size of the root dbspace depends on whether you plan to store the physical log, logical logs, and temporary tables in the root dbspace or in another dbspace. The root dbspace must be large enough for the minimum size configuration during disk initialization.

Recommendation: Set up the system with a small log size (for example, three 1000-kilobyte log files, or 3000 kilobytes for the total log size). After setup is complete, create new dbspaces, move and resize the logical-log files, and drop the original logs in the root dbspace. Then move the physical log to another dbspace. This procedure minimizes the impact of the logs in the root dbspace because:

- A large amount of space is not left unused in the root dbspace after you move the logs.
- The logs do not contend for space and I/O on the same disk as the root dbspace.

For details on how to move the logs, see "Moving a Logical-Log File to Another Dbspace" on page 14-13 and "Changing the Physical-Log Location and Size" on page 16-1.

After the database server is initialized, you cannot change the size of the root dbspace without reinitializing the database server and destroying all data on the disk. If you change the value of the ROOTSIZE configuration parameter after the database server is initialized, the change will not be effective until the database server is reinitialized.

Physical and Logical Logs

The value stored in the ONCONFIG parameter PHYSFILE defines the size of your physical log when the database server is initially created. After you use the **oninit -i** command to initialize the disk space and bring the database server online, use the **onparams** utility to change the physical log location and size. Advice on sizing your physical log is contained in "Size and Location of the Physical Log" on page 15-2.

To calculate the size of the logical-log files, multiply the value of the ONCONFIG parameter LOGSIZE by the number of logical-log files. For advice on sizing your logical log, see "Estimating the Size and Number of Log Files" on page 14-1.

Temporary Tables

Analyze end-user applications to estimate the amount of disk space that the database server might require for temporary tables. Try to estimate how many of these statements are to run concurrently. The space occupied by the rows and columns that are returned provides a good basis for estimating the amount of space required.

The largest temporary table that the database server creates during a warm restore is equal to the size of your logical log. You calculate the size of your logical log by adding the sizes of all logical-log files.

You must also analyze end-user applications to estimate the amount of disk space that the database server might require for explicit temporary tables.

For more information, including a list of statements that require temporary space, see "Temporary Tables" on page 9-25.

Critical Data

Recommendation: Do not store databases and tables in the root dbspace. Mirror the root dbspace and other dbspaces that contain critical data such as the physical log and logical logs. Estimate the amount of disk space, if any, that you need to allocate for tables stored in the root dbspace.

Extra Space

Allow extra space in the root dbspace for the system databases to grow, for the extended reserved pages, and ample free space. The number of extended reserved pages depends on the number of primary chunks, mirror chunks, logical-log files, and storage spaces in the database server.

Amount of Space That Databases Require

The amount of additional disk space required for the database server data storage depends on the needs of your end users, plus overhead and growth. Every application that your end users run has different storage requirements. The following list suggests some of the steps that you can take to calculate the amount of disk space to allocate (beyond the root dbspace):

- Decide how many databases and tables you need to store. Calculate the amount of space required for each one.
- Calculate a growth rate for each table and assign some amount of disk space to each table to accommodate growth.
- Decide which databases and tables you want to mirror.

For instructions about calculating the size of your tables, refer to your *IBM Informix Dynamic Server Performance Guide*.

Disk-Layout Guidelines

The following are typical goals for efficient disk layout:

- Limiting disk-head movement
- Reducing disk contention
- Balancing the load
- Maximizing availability

You must make some trade-offs among these goals when you design your disk layout. For example, separating the system catalog tables, the logical log, and the physical log can help reduce contention for these resources. However, this action can also increase the chances that you have to perform a system restore. For detailed disk-layout guidelines, see the *IBM Informix Dynamic Server Performance Guide*.

Dbspace and Chunk Guidelines

This section lists some general strategies for disk layout that do not require any information about the characteristics of a particular database:

- Associate disk partitions with chunks and allocate at least one *additional* chunk for the root dbspace.

A disk that is already partitioned might require the use of offsets. For details, see "Allocating Raw Disk Space on UNIX" on page 10-3.

Tip: With the 4-terabyte maximum size of a chunk, you can avoid partitioning by assigning a chunk per disk drive.

- Mirror critical dbspaces: the root dbspace, the dbspaces that contain the physical log and the logical-log files. Also mirror high-use databases and tables.

  You specify mirroring at the dbspace level. Mirroring is either on or off for all chunks belonging to a dbspace. Locate the primary and the mirrored dbspaces on different disks. Ideally, different controllers handle the different disks.
- Spread temporary tables and sort files across multiple disks.

  To define several dbspaces for temporary tables and sort files, use **onspaces -t**. When you place these dbspaces on different disks and list them in the DBSPACETEMP configuration parameter, you can spread the I/O associated with temporary tables and sort files across multiple disks. For information on using the DBSPACETEMP configuration parameter or environment variable, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.
- Keep the physical log in the root dbspace but move the logical logs from the root dbspace. However, if you plan to store the system catalogs in the root dbspace, move the physical log to another dbspace.

  For advice on where to store your logs, see "Specifying the Location of the Physical Log" on page 15-3 and "Location of Logical-Log Files" on page 13-1. Also see "Moving a Logical-Log File to Another Dbspace" on page 14-13 and "Changing the Physical-Log Location and Size" on page 16-1.
- To improve backup and restore performance:
  - Cluster system catalogs with the data that they track.
  - If you use ON–Bar to perform parallel backups to a high-speed tape drive, store the databases in several small dbspaces.

    For additional performance recommendations, see the *IBM Informix Backup and Restore Guide*.

Table-Location Guidelines

This section lists some strategies for optimizing the disk layout, given certain characteristics about the tables in a database. You can implement many of these strategies with a higher degree of control using table fragmentation:

- Isolate high-use tables on a separate disk.

  To isolate a high-use table on its own disk device, assign the device to a chunk, and assign the same chunk to a dbspace. Finally, place the frequently used table in the dbspace just created using the IN *dbspace* option of CREATE TABLE.

  To display the level of I/O operations against each chunk, run the **onstat -g iof** option.
- Fragment high-use tables over multiple disks.
- Group related tables in a dbspace.

  If a device that contains a dbspace fails, all tables in that dbspace are inaccessible. However, tables in other dbspaces remain accessible. Although you must perform a cold restore if a dbspace that contains critical information fails, you need only perform a warm restore if a noncritical dbspace fails.
- Place high-use tables on the middle partition of a disk.
- Optimize table extent sizes.

For more information, see the chapter on table performance considerations in your *IBM Informix Performance Guide*. For information on **onstat** options, see the *IBM Informix Administrator's Reference*.

Sample Disk Layouts

When setting out to organize disk space, the database server administrator usually has one or more of the following objectives in mind:

- High performance
- High availability
- Ease and frequency of backup and restore

Meeting any one of these objectives has trade-offs. For example, configuring your system for high performance usually results in taking risks regarding the availability of data. The sections that follow present an example in which the database server administrator must make disk-layout choices given limited disk resources. These sections describe two different disk-layout solutions. The first solution represents a performance optimization, and the second solution represents an availability-and-restore optimization.

The setting for the sample disk layouts is a fictitious sporting goods database that uses the structure (but not the volume) of the **stores_demo** database. In this example, the database server is configured to handle approximately 350 users and 3 gigabytes of data. The disk space resources are shown in the following table.

Disk Drive	Size of Drive	High Performance
Disk 1	2.5 gigabytes	No
Disk 2	3 gigabytes	Yes
Disk 3	2 gigabytes	Yes
Disk 4	1.5 gigabytes	No

The database includes two large tables: **cust_calls** and **items**. Assume that both of these tables contain more than 1,000,000 rows. The **cust_calls** table represents a record of all customer calls made to the distributor. The items table contains a line item of every order that the distributor ever shipped.

The database includes two high-use tables: **items** and **orders**. Both of these tables are subject to constant access from users around the country.

The remaining tables are low-volume tables that the database server uses to look up data such as postal code or manufacturer.

Table Name	Maximum Size	Access Rate
cust_calls	2.5 gigabytes	Low
items	0.5 gigabytes	High
orders	50 megabytes	High
customers	50 megabytes	Low
stock	50 megabytes	Low
catalog	50 megabytes	Low
manufact	50 megabytes	Low

Table Name	Maximum Size	Access Rate
state	50 megabytes	Low
call_type	50 megabytes	Low

Sample Layout When Performance Is Highest Priority

Figure 9-18 shows a disk layout optimized for performance. This disk layout uses the following strategies to improve performance:

- Migration of the logical log from the rootdbs dbspace to a dbspace on a separate disk

  This strategy separates the logical log and the physical log and reduces contention for the root dbspace.

- Location of the two tables that undergo the highest use in dbspaces on separate disks

  Neither of these disks stores the logical log or the physical log. Ideally you could store each of the **items** and **orders** tables on a separate high-performance disk. However, in the present scenario, this strategy is not possible because one of the high-performance disks is needed to store the very large **cust_calls** table (the other two disks are too small for this task).

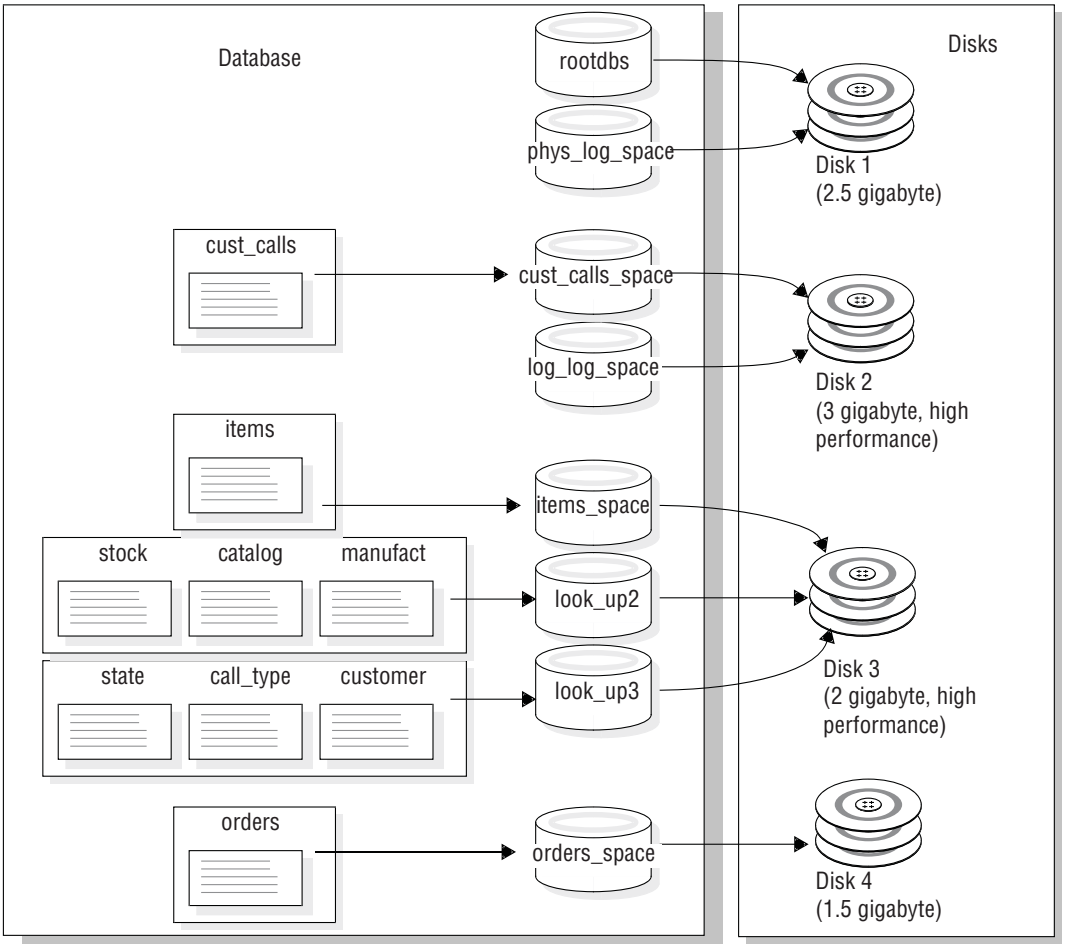


Figure 9-18. Disk Layout Optimized for Performance

Sample Layout When Availability Is Highest Priority

The weakness of the previous disk layout is that if either Disk 1 or Disk 2 fails, the whole database server goes down until you restore the dbspaces on these disks from backups. In other words, the disk layout is poor with respect to availability.

An alternative disk layout that optimizes for availability and involves mirroring is shown in Figure 9-19. This layout mirrors all the critical data spaces (the system catalog tables, the physical log, and the logical log) to a separate disk. Ideally you could separate the logical log and physical log (as in the previous layout) and mirror each disk to its own mirror disk. However, in this scenario, the required number of disks does not exist; therefore, the logical log and the physical log both reside in the root dbspace.

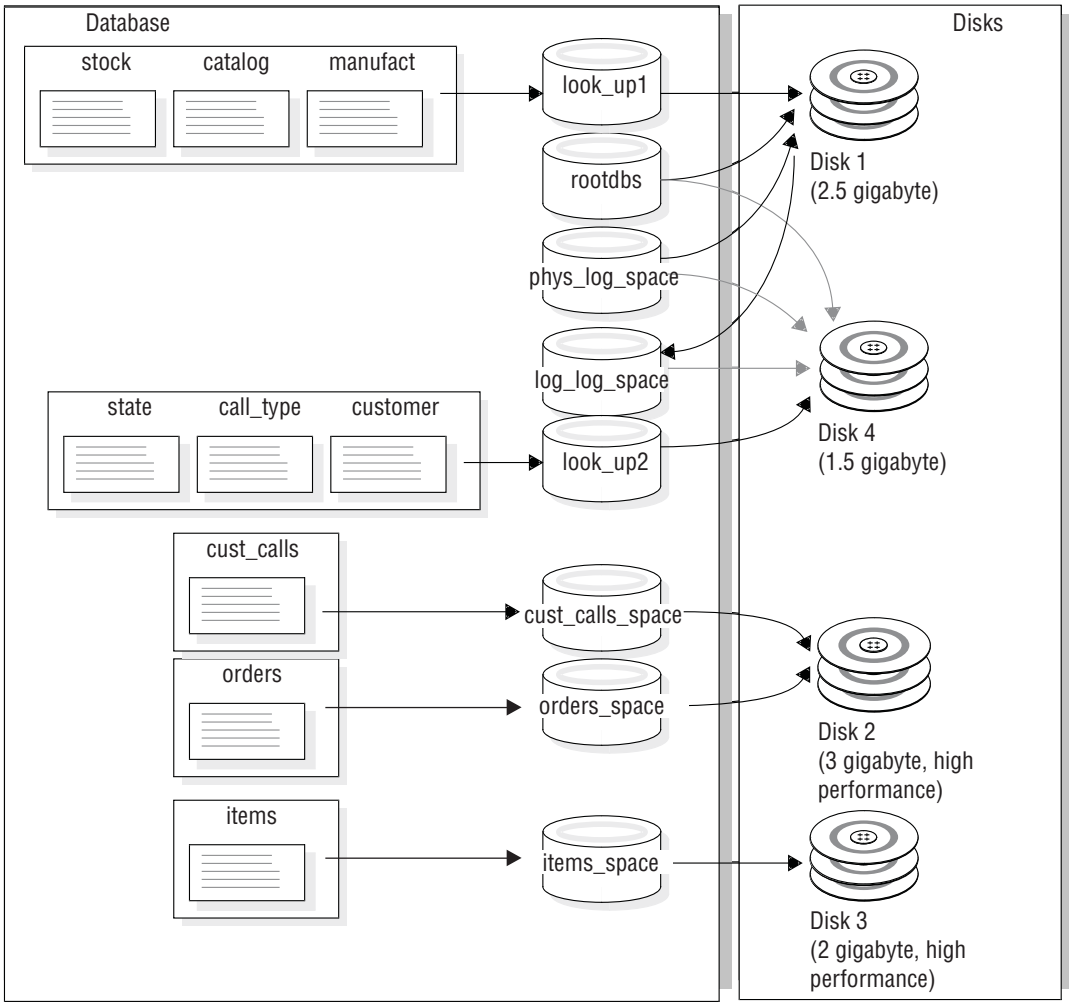


Figure 9-19. Disk Layout Optimized for Availability

Logical-Volume Manager

A logical-volume manager (LVM) is a utility that allows you to manage your disk space through user-defined logical volumes.

Many computer manufacturers ship their computers with a proprietary LVM. You can use the database server to store and retrieve data on disks that are managed

by most proprietary LVMs. Logical-volume managers provide some advantages and some disadvantages, as discussed in the remainder of this section.

Most LVMs can manage multiple gigabytes of disk space. The database server chunks are limited to a size of 4 terabytes, and this size can be attained only when the chunk being allocated has an offset of zero. Consequently, you should limit the size of any volumes to be allocated as chunks to a size of 4 terabytes.

Because LVMs allow you to partition a disk drive into multiple volumes, you can control where data is placed on a given disk. You can improve performance by defining a volume that consists of the middle-most cylinders of a disk drive and placing high-use tables in that volume. (Technically, you do not place a table directly in a volume. You must first allocate a chunk as a volume, then assign the chunk to a dbspace, and finally place the table in the dbspace. For more information, see "Control of Where Data Is Stored" on page 9-8.)

Tip: If you choose to use large disk drives, you can assign a chunk to one drive and eliminate the need to partition the disk.

You can also improve performance by using a logical volume manager to define a volume that spreads across multiple disks and then placing a table in that volume.

Many logical volume managers also allow a degree of flexibility that standard operating-system format utilities do not. One such feature is the ability to reposition logical volumes after you define them. Thus getting the layout of your disk space right the first time is not so critical as with operating-system format utilities.

LVMs often provide operating-system-level mirroring facilities. For more information, see "Alternatives to Mirroring" on page 17-2.

Chapter 10. Managing Disk Space

In This Chapter

This chapter provides the instructions on managing effectively the disk spaces and data that the database server controls. It assumes you are familiar with the terms and concepts contained in Chapter 9, "Data Storage," on page 9-1.

You can use the following utilities to manage storage spaces:

- **onspaces**
- OpenAdmin Tool
- ISA

This chapter covers the following topics:

- Allocating disk space
- Specifying chunk names, and the maximum size and number of chunks and storage spaces
- Backing up after you change the physical schema
- Monitor storage spaces
- Creating and dropping dbspaces, blobspaces, and sbspaces
- Renaming dbspaces
- Managing dbspace Partitions
- Adding, and dropping chunks from dbspaces, blobspaces, and sbspaces
- Creating and dropping extspaces
- Skipping inaccessible fragments
- Monitoring disk space usage
- Monitoring simple-large-object data and smart-large-object data

Your *IBM Informix Dynamic Server Performance Guide* also contains information about managing disk space. In particular, it describes how to eliminate interleaved extents, how to reclaim space in an empty extent, and how to improve disk I/O.

For information on using SQL administration API commands instead of some **onspaces** commands, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1 and the *IBM Informix Dynamic Server Administrator's Reference*.

Allocating Disk Space

This section explains how to allocate disk space for the database server. Read the following sections before you allocate disk space:

- "Unbuffered or Buffered Disk Access on UNIX" on page 9-3
- "Amount of Disk Space Needed to Store Data" on page 9-31
- "Disk-Layout Guidelines" on page 9-33

Before you can create a storage space or chunk, or mirror an existing storage space, you must allocate disk space for the chunk file. You can allocate either an empty file or a portion of raw disk for database server disk space.

UNIX Only:

On UNIX, if you allocate raw disk space, you should use the UNIX **ln** command to create a link between the character-special device name and another filename. For more information on this topic, see "Creating Symbolic Links to Raw Devices (UNIX)" on page 10-4.

Using a UNIX file and its inherent operating-system interface for database server disk space also is referred to as using *cooked space.*

Windows Only:

On Windows, you should use NTFS files for database server disk space. For more information on this recommendation, see "Unbuffered or Buffered Disk Access on UNIX" on page 9-3.

You can balance chunks over disks and controllers. Placing multiple chunks on a single disk can improve throughput.

Specifying an Offset

When you allocate a chunk of disk space to the database server, specify an offset for one of the following two purposes:

- To prevent the database server from overwriting the partition information
- To define multiple chunks on a partition, disk device, or cooked file

The maximum value for the offset is 4 terabytes.

Many computer systems and some disk-drive manufacturers keep information for a physical disk drive on the drive itself. This information is sometimes referred to as a *volume table of contents* (VTOC) or disk label. The VTOC is commonly stored on the first track of the drive. A table of alternate sectors and bad-sector mappings (also called a *revectoring table*) might also be stored on the first track.

If you plan to allocate partitions at the start of a disk, you might need to use offsets to prevent the database server from overwriting critical information required by the operating system. For the exact offset required, refer to your disk-drive manuals.

Warning: If you are running two or more instances of the database server, be extremely careful not to define chunks that overlap. Overlapping chunks can cause the database server to overwrite data in one chunk with unrelated data from an overlapping chunk. This overwrite effectively destroys overlapping data.

Specifying an Offset for the Initial Chunk of Root Dbspace

For the initial chunk of root dbspace and its mirror, if it has one, specify the offsets with the ROOTOFFSET and MIRROROFFSET parameters, respectively. For more information, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

Specifying an Offset for Additional Chunks

To specify an offset for additional chunks of database server space, you must supply the offset as a parameter when you assign the space to the database server. For more information, see "Creating a Dbspace that Uses the Default Page Size" on page 10-7.

Using Offsets to Create Multiple Chunks

You can create multiple chunks from a disk partition, disk device, or file, by specifying offsets and assigning chunks that are smaller than the total space available. The offset specifies the beginning location of a chunk. The database server determines the location of the last byte of the chunk by adding the size of the chunk to the offset.

For the first chunk, assign any initial offset, if necessary, and specify the size as an amount that is less than the total size of the allocated disk space. For each additional chunk, specify the offset to include the sizes of all previously assigned chunks, plus the initial offset, and assign a size that is less than or equal to the amount of space remaining in the allocation.

Allocating Cooked File Spaces on UNIX

The following procedure shows an example of allocating disk space for a cooked file, called **usr/data/my_chunk**, on UNIX.

To allocate cooked file space

1. Log in as user **informix**:

   ```
   su informix
   ```

2. Change directories to the directory where the cooked space will reside:

   ```
   cd /usr/data
   ```

3. Create your chunk by concatenating null to the filename that the database server will use for disk space:

   ```
   cat /dev/null > my_chunk
   ```

4. Set the file permissions to 660 (rw-rw----):

   ```
   chmod 660 my_chunk
   ```

5. You must set both group and owner of the file to **informix**:

   ```
   ls -l my_chunk  -rw-rw----
     1  informix   informix
     0  Oct 12 13:43 my_chunk
   ```

6. Use **onspaces** to create the storage space or chunk.

For information on how to create a storage space using the file you have allocated, refer to “Creating a Dbspace that Uses the Default Page Size” on page 10-7, “Creating a Blobspace” on page 10-20, and “ Creating an Sbspace” on page 10-23.

Allocating Raw Disk Space on UNIX

To allocate raw space, you must have a disk partition available that is dedicated to raw space. To create raw disk space, you can either repartition your disks or unmount an existing file system. Back up any files before you unmount the device.

To allocate raw disk space

1. Create and install a raw device.

   For specific instructions on how to allocate raw disk space on UNIX, see your operating-system documentation and “Unbuffered or Buffered Disk Access on UNIX” on page 9-3.

2. Change the ownership and permissions of the character-special devices to **informix**.

The filename of the character-special device usually begins with the letter *r*. For the procedure, see steps 4 on page 10-3 and 5 on page 10-3 in "Allocating Cooked File Spaces on UNIX" on page 10-3.

3. Verify that the operating-system permissions on the character-special devices are `crw-rw----`.
4. Create a symbolic link between the character-special device name and another filename with the UNIX link command, **ln -s**. For details, see "Creating Symbolic Links to Raw Devices (UNIX)."

Warning: After you create the raw device that the database server uses for disk space, do not create file systems on the same raw device that you allocate for the database server disk space. Also, do not use the same raw device as swap space that you allocate for the database server disk space.

Creating Symbolic Links to Raw Devices (UNIX)

Use symbolic links to assign standard device names and to point to the device. To create a link between the character-special device name and another filename, use the UNIX link command (usually **ln)**. To verify that both the devices and the links exist, execute the UNIX command **ls -l** (**ls -lg** on BSD) on your device directory. The following example shows links to raw devices. If your operating system does not support symbolic links, hard links also work.

```
ln -s /dev/rxy0h /dev/my_root # orig_device link to symbolic_name
ln -s /dev/rxy0a /dev/raw_dev2
ls -l
crw-rw---  /dev/rxy0h
crw-rw---  /dev/rxy0a
lrwxrwxrwx /dev/my_root@->/dev/rxy0h
lrwxrwxrwx /dev/raw_dev2@->/dev/rxy0a
```

Why use symbolic links? If you create chunks on a raw device and that device fails, you cannot restore from a backup until you replace the raw device and use the same path name. All chunks that were accessible at the time of the last backup must be accessible when you perform the restore.

Symbolic links simplify recovery from disk failure and enable you to replace quickly the disk where the chunk is located. You can replace a failed device with another device, link the new device path name to the same filename that you previously created for the failed device, and restore the data. You do not need to wait for the original device to be repaired.

Allocating NTFS File Space on Windows

On Windows, the database server uses NTFS files by default. The NTFS file system allows you to use standard filenames for unbuffered files.

To allocate NTFS file space for database server disk space or mirrored space, the first step is to create a null (zero bytes) file.

To allocate NTFS file space for a dbspace, blobspace, or sbspace

1. Log in as a member of the **Informix-Admin** group.
2. Open an MS-DOS command shell.
3. Change to the directory where you want to allocate the space, as in the following example:

   ```
   c:> cd \usr\data
   ```

4. Create a null file with the following command:

   ```
   c:> copy nul my_chunk
   ```

5. If you want to verify that the file has been created, use the **dir** command to do so.

After you have allocated the file space, you can create the dbspace or other storage space as you normally would, using **onspaces**. For information on how to create a dbspace or a blobspace, refer to "Creating a Dbspace that Uses the Default Page Size" on page 10-7 and "Creating a Blobspace" on page 10-20.

Allocating Raw Disk Space on Windows

You can configure raw disk space on Windows as a logical drive or physical drive. To find the drive letter or disk number, run the **Disk Administrator**. If the drives need to be striped (multiple physical disks combined into one logical disk), only logical drive specification would work.

You must be a member of the **Informix-Admin** group when you create a storage space or add a chunk. The raw disk space can be formatted or unformatted disk space.

To specify a logical drive

1. Assign a drive letter to the disk partition.
2. Specify the following value for ROOTDBS in the ONCONFIG file:

   ```
   \\.\drive_letter
   ```

3. To create a storage space or add a chunk, specify the logical drive partition.

   This example adds a chunk of 5000 kilobytes on the **e:** drive, at an offset of 5200 kilobytes, to dbspace **dpspc3.**

   ```
   onspaces -a dbspc3 \\.\e: -o 5200 -s 5000
   ```

To specify a physical drive

1. If the disk partition has *not* been assigned a drive letter, specify the following value for ROOTDBS in the ONCONFIG file:

   ```
   \\.\PhysicalDrive<number>
   ```

2. To create a storage space or add a chunk, specify the physical drive partition.

   This example adds a chunk of 5000 kilobytes on **PhysicalDrive0**, at an offset of 5200 kilobytes, to dbspace **dpspc3.**

   ```
   onspaces -a dbspc3 \\.\PhysicalDrive0 : -o 5200 -s 5000
   ```

Warning: If you allocate a formatted drive or disk partition as raw disk space and it contains data, the database server overwrites the data when it begins to use the disk space. You must ensure that any data on raw disk space is expendable before you allocate the disk space to the database server.

Specifying Names for Storage Spaces and Chunks

Chunk names follow the same rules as storage-space names. Specify an explicit path name for a storage space or chunk as follows:

- If you are using raw disks on UNIX, you should use a linked path name. (See "Creating Symbolic Links to Raw Devices (UNIX)" on page 10-4.)
- If you are using raw disks on Windows, the path name takes the following form, where *x* specifies the disk drive or partition:

\\.*x*:

- If you are using a file for database server disk space, the path name is the complete path and filename.

Use these naming rules when you create storage spaces or add a chunk. The filename must have the following characteristics:

- Be unique and not exceed 128 bytes
- Begin with a letter or underscore
- Contain only letters, digits, underscores, or **$** characters

The name is case insensitive unless you use quotes around it. By default, the database server converts uppercase characters in the name to lowercase. If you want to use uppercase in names, put quotes around them and set the **DELIMIDENT** environment variable to `ON`.

Specifying the Maximum Size of Chunks

On most platforms, the maximum chunk size is 4 terabytes, but on other platforms, the maximum chunk size is 8 terabytes. To determine which chunk size your platform supports refer to your machine notes file. If you do not run **onmode -BC**, then the maximum chunk size is 2 GB.

Specifying the Maximum Number of Chunks and Storage Spaces

You can specify a maximum of 32,766 chunks for a storage space, and a maximum of 32,766 storage spaces on the database server system. The storage spaces can be any combination of dbspaces, blobspaces, and sbspaces.

Considering all limits that can apply to the size of an instance of the database server, the maximum size of an instance is approximately 8 petabytes.

You must run **onmode -BC** to enable the maximum number of chunks and storage spaces.

Backing Up After You Change the Physical Schema

You must perform a level-0 backup of the root dbspace and the modified storage spaces to ensure that you can restore the data when you:

- Add or drop mirroring
- Drop a logical-log file
- Change the size or location of the physical log
- Change your storage-manager configuration
- Add, move, or drop a dbspace, blobspace, or sbspace
- Add, move, or drop a chunk to a dbspace, blobspace, or sbspace

Important: When you add a new logical log, you no longer need to perform a level-0 backup of the root dbspace and modified dbspace to *use* the new logical log. However, you should perform the level-0 backup to prevent level-1 and level-2 backups from failing.

You must perform a level-0 backup of the modified storage spaces to ensure that you can restore the unlogged data before you switch to a logging table type:

- When you convert a nonlogging database to a logging database
- When you convert a RAW table to standard

Monitoring Storage Spaces

You can monitor the status of storage spaces and configure how you are notified when a storage space becomes full.

When a storage space or partition becomes full, a message is shown in the online message log file.

You can configure alarms that are triggered when storage spaces become full with the STORAGE_FULL_ALARM configuration parameter. You can specify how often alarms are sent and the minimum severity level of alarms to be sent. By default, the alarm interval is 600 seconds and the alarm severity level is 3. For more information on the STORAGE_FULL_ALARM configuration parameters and event alarms, see the *IBM Informix Administrator's Reference*.

You can use the Dynamic Server Scheduler to set up a task that automatically monitors the status of storage spaces. The properties of the task define the information that Scheduler collects and specifies how frequently the task runs. For example, you could define a task to monitor storage spaces every hour, five days a week. For more information, see Chapter 30, "The Scheduler," on page 30-1 and "Setting Up Tasks" on page 30-2.

Managing Dbspaces

This section contains information on creating standard and temporary dbspaces with and without the default page size, specifying the first and next extent sizes for the tblspace **tblspace** in a dbspace when you create the dbspace, and adding a chunk to a dbspace or blobspace.

For information on monitoring dbspaces, see "Monitoring Storage Spaces"

For more information about using the ON-Monitor and **onspaces** utilities, see the *IBM Informix Dynamic Server Administrator's Reference*.

Creating a Dbspace that Uses the Default Page Size

This section explains how to use **onspaces** to create a standard dbspace and a temporary dbspace. For information on creating a dbspace with a non-default page size, see "Creating a Dbspace with a Non-Default Page Size" on page 10-10.

For information on using ISA to create a dbspace, see the ISA online help.

Any newly added dbspace (and its mirror, if one exists) is available immediately. If you are using mirroring, you can mirror the dbspace when you create it. Mirroring takes effect immediately.

To create a standard dbspace using onspaces

1. On UNIX, you must be logged in as user **informix** or **root** to create a dbspace.
   On Windows, users in the **Informix-Admin** group can create a dbspace.

2. Ensure that the database server is in online, administration, or quiescent mode.
3. Allocate disk space for the dbspace, as described in "Allocating Disk Space" on page 10-1.
4. To create a dbspace, use the **onspaces -c -d** options.

   Kilobytes is the default unit for the **-s** *size* and **-o offset** options. To convert kilobytes to megabytes, multiply the unit by 1024 (for example, `10 MB = 10 * 1024 KB`).

   See "Creating a Dbspace with a Non-Default Page Size" on page 10-10 for information on additional **onspaces** options if you are creating a dbspace with a non-default page size.
5. If you do not want to specify the first and next extent sizes for the tblspace **tblspace** in a dbspace, go to step 6.

   If you want to specify the first and next extent sizes for the tblspace **tblspace** in a dbspace, see additional information in "Specifying the First and Next Extent Sizes for the tblspace **tblspace**" on page 10-9.
6. After you create the dbspace, you must perform a level-0 backup of the root dbspace and the new dbspace.

The following example shows how to create a 10-megabyte mirrored dbspace, **dbspce1**, with an offset of 5000 kilobytes for both the primary and mirror chunks, using raw disk space on UNIX:

```
onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10240 -m /dev/raw_dev2 5000
```

The following example shows how to create a 5-megabyte dbspace, **dbspc3**, with an offset of 200 kilobytes, from raw disk space (drive `e:`) on Windows:

```
onspaces -c -d dbspc3 \\.\e: -o 200 -s 5120
```

For more information on creating a dbspace with **onspaces**, see "Dbspaces" on page 9-8. and information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

To create a dbspace with ON-Monitor (UNIX)

1. Select the **Dbspaces > Create** option.
2. Enter the name of the new dbspace in the field **Dbspace Name**.
3. If you want to create a mirror for the initial dbspace chunk, enter `Y` in the **Mirror** field.

   Otherwise, enter `N`.
4. If the dbspace that you are creating is a temporary dbspace, enter `Y` in the **Temp** field.

   Otherwise, enter `N`.
5. If you are specifying a page size for a standard dbspace, enter the size in kilobytes in the **Page Size** field. The size must be a multiple of the page size of the root dbspace. For more information on specifying page sizes, see "Creating a Dbspace with a Non-Default Page Size" on page 10-10.

   All tables, indexes, and other objects within the dbspace will use pages of the specified size.
6. Enter the full path name for the initial primary chunk of the dbspace in the **Full Pathname** field of the primary-chunk section.
7. Specify an offset in the **Offset** field.
8. Enter the size of the chunk, in kilobytes, in the **Size** field.

9. If you are mirroring this dbspace, enter the full path name, size, and optional offset of the mirror chunk in the mirror-chunk section of the screen.

For more information, refer to the ON–Monitor chapter in the *IBM Informix Administrator's Reference*.

Specifying the First and Next Extent Sizes for the tblspace tblspace

Specify first and next extent sizes if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you need to place the tblspace **tblspace** extents in non-primary chunks. (A *primary chunk* is the initial chunk in a dbspace.)

You can choose to specify the first extent size, the next extent size, both the first and the next extent size, or neither extent size. If you do not specify first or next extent sizes for the tblspace **tblspace**, Dynamic Server uses the existing default extent sizes.

You can use the TBLTBLFIRST and TBLTBLNEXT configuration parameters to specify the first and next extent sizes for the tblspace **tblspace** in the root dbspace that is created when the server is initialized.

You can use the **onspaces** utility to specify the first and next extent sizes for the tblspace **tblspace** in non-root dbspaces.

You can only specify the first and next extent sizes when you create dbspace. You cannot alter the specification of the first and next extent sizes after the creation of the dbspace. In addition, you cannot specify extent sizes for temporary dbspaces, sbspaces, blobspaces, or external spaces. You cannot alter the specification of the first and next extents sizes after the creation of the dbspace.

To specify the first and next extent sizes

1. Determine the total number of pages needed in the tblspace **tblspace**. The number of pages is equal to the sum of the number of tables, detached indexes, and table fragments likely to reside in the dbspace plus one page for the tblspace **tblspace**.
2. Calculate the number of kilobytes needed for the number of pages. This number will depend on the number of kilobytes to a page on the system.
3. Determine the space management needs on your system by considering the importance of having all of the extents for the tblspace **tblspace** allocated during dbspace creation and whether the extents should be allocated contiguously. The more important these issues are, the larger the first extent size should be. If you are less concerned with having non-contiguous extents, possibly in secondary chunks, then the first and next extent sizes can be smaller.
4. Specify the extent size as follows:
   - If the space requirement is for the root dbspace, specify the first extent size in the TBLTBLFIRST configuration parameter and the next extent size in the TBLTBLNEXT configuration parameter. Then initialize the database server instance.
   - If the space requirement is for a non-root dbspace, indicate the first and next extent sizes on the command line using the **onspaces** utility to create the dbspace.

Extent sizes must be in kilobytes and must be multiples of the page size. When you specify first and next extent sizes, follow these guidelines:

Type of Extent	Minimum Size	Maximum Size
First extent in a non-root dbspace	The equivalent of 50 pages, specified in kilobytes. This is the system default. For example, for a two-kilobyte page system, the minimum length is 100.	The size of the initial chunk, minus the space needed for any system objects such as the reserved pages, the database tblspace, and the physical and logical logs.
First extent in a root dbspace	The equivalent of 250 pages specified in kilobytes. This is the system default.	The size of the initial chunk, minus the space needed for any system objects such as the reserved pages, the database tblspace, and the physical and logical logs.
Next Extent	Four times the disk-page size on the system. The default is 50 pages on any type of dbspace.	The maximum chunk size minus three pages.

You use the following **onspaces** utility **-ef** and **-en** options to specify the first and next extent sizes for the tblspace **tblspace** in non-root dbspaces:

- First extent size: **-ef** *size_in_kbytes*
- Next extent size: **-en** *size_in_kbytes*

For example, you can specify:

```
onspaces -c -d dbspace1 -p /usr/data/dbspace1 -o 0 -s 1000000 -e 2000 -n 1000
```

You can use **Oncheck -pt** and **oncheck -pT** to show the first and next extent sizes of a tblspace **tblspace**.

If data replication is being used and a dbspace is created on the primary database server, the first and next extent sizes are passed to the secondary database server via the ADDCHK log record.

For more information on the **onspaces** utility, **oncheck** commands, and specifying the first and next extent sizes for the tblspace **tblspace**, see the *IBM Informix Administrator's Reference*.

Creating a Dbspace with a Non-Default Page Size

You can specify a page size for a standard or temporary dbspace if you want a longer key length than is available for the default page size. The root dbspace is the default page size. If you want to specify a page size, the size must be an integral multiple of the default page size, and cannot be greater than 16 kilobytes.

For systems with sufficient storage, the performance advantages of a larger page size include:

- Reduced depth of b-tree indexes, even for smaller index keys.
- Decreased checkpoint time, which typically occurs with larger page sizes.

Additional performance advantages occur because you can:

- Group on the same page long rows that currently span multiple pages of the default page size.
- Define a different page size for temporary tables, so the temporary tables have a separate buffer pool.

You can use the BUFFERPOOL configuration parameter to create a buffer pool that corresponds to the page size of the dbspace. (You might want to do this to implement a form of ″private buffer pool.″)

A table can be in one dbspace and the index for that table can be in another dbspace. The page size for these partitions can be different.

If you want to specify the page size for the dbspace, perform these tasks:

1. Use the **onmode -BC** command to enable the large chunk mode if this mode is not enabled. By default, when Dynamic Server is first initialized or restarted, Dynamic Server starts with the large chunk mode enabled. For information on the **onmode** utility, see the *IBM Informix Dynamic Server Administrator's Reference*.
2. Create a buffer pool that corresponds to the page size of the dbspace. You can use the **onparams** utility or the BUFFERPOOL configuration parameter. You should do this before you create the dbspace.

   If you create a dbspace with a page size that does not have a corresponding buffer pool, Dynamic Server automatically creates a buffer pool using the default parameters defined in the ONCONFIG configuration file.

   You cannot have multiple buffer pools with the same page size.

   For more information, see "Creating a Dbspace with a Non-Default Page Size" on page 10-10.
3. Define the page size of the dbspace when you create the dbspace. You can use the **onspaces** utility or ON-Monitor. For more information, see "Defining the Page Size" on page 10-15.

For example, if you create a dbspace with a page size of 6 kilobytes, you must create a buffer pool with a size of 6 kilobytes. If you do not specify a page size for the new buffer pool, Dynamic Server uses the operating system default page size (4 kilobytes on Windows and 2 kilobytes on most UNIX platforms) as the default page size for the buffer pool.

Note: If you use non-default page sizes, you might need to increase the size of your physical log. If you perform many updates to non-default pages you might need a 150 to 200 percent increase of the physical log size. Some experimentation might be needed to tune the physical log. You can adjust the size of the physical log as necessary according to how frequently the filling of the physical log triggers checkpoints.

Creating a Buffer Pool for the Non-Default Page Size

When you create a buffer pool, you can use the BUFFERPOOL configuration parameter or the **onparams** utility to define information about the buffer pool including its size, the number of LRUS in the buffer pool, the number of buffers in the buffer pool, and **lru_min_dirty** and **lru_max_dirty** values.

You specify this information using the BUFFERPOOL configuration parameter.

The BUFFERPOOL configuration parameter consists of two lines in the **onconfig.std** file.

On UNIX systems, the lines are:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=2K,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

On Windows systems, the lines are:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=4K,buffers=10000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

The top line specifies the default values that are used if you create a dbspace with a page size that does not have a corresponding buffer pool that was created when the database server started. The line below the default line specifies the database server default values for the buffer pool. These values are based on the default page size of the database server.

When you add a dbspace with a different page size with the **onspaces** utility or when you add a new buffer pool with the **onparams** utility, a new line is added to the BUFFERPOOL configuration parameter in the ONCONFIG file. The page size for each buffer pool must be a multiple of the default page size for your operating system. The following example shows a third BUFFERPOOL line that was added to the ONCONFIG file:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=2K,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
BUFFERPOOL size=6K,buffers=3000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

The fields in the BUFFERPOOL lines are not case sensitive, so you can specify **lrus** or **Lrus** or **LRUS**). These fields can appear in any order.

If you do not specify a page size for a new buffer pool, Dynamic Server uses the operating system default page size (4 kilobytes on Windows and 2 kilobytes on most UNIX platforms) as the default page size for the buffer pool.

If the value of **buffers** is zero (0) or if the value of **buffers** is missing in the BUFFERPOOL configuration parameter, Dynamic Server will not create the buffer pool of the specified page size.

Note: Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters prior to Version 10.0 is now specified using the BUFFERPOOL configuration parameter. Information you enter using the BUFFERPOOL configuration parameter supersedes any information previously specified with the deprecated parameters. For more information on the deprecated parameters, see the appendix that contains information on discontinued configuration parameters that is in *IBM Informix Dynamic Server Administrator's Reference*.

The table below provides an explanation of the values that you specify using the BUFFERPOOL configuration parameter or the **onparams** utility.

Field	Explanation	Range of Values
size	Specifies the number of kilobytes, followed by the suffix `K`, in a page.	The size can vary from 2K or 4K to 16 K. `2K` is the default.

Field	Explanation	Range of Values
buffers	Specifies the number of buffers for the page size. This is the maximum number of shared-memory buffers that the database server user threads have available for disk I/O on behalf of client applications. The number of buffers that the database server requires depends on the applications. For example, if the database server accesses 15 percent of the application data 90 percent of the time, you need to allocate enough buffers to hold that 15 percent. Increasing the number of buffers can improve system performance. The percentage of physical memory that you need for buffer space depends on the amount of memory available on your computer and the amount of memory used for other applications. For systems with a large amount of available physical memory (4 gigabytes or more), buffer space can be as much as 90 percent of physical memory. For systems with smaller amounts of available physical memory, buffer space can range from 20 to 25 percent of physical memory. You should calculate all other shared-memory parameters after you set buffer space (`buffers` **system_page_size*).	For 32-bit platform on UNIX • with page size equal to 2048 bytes: 100 through 1,843,200 buffers (`1843200 = 1800 * 1024` • with page size equal to 4096 bytes: 100 through 921,600 buffers (`921,600 = ((1800 * 1024)/4096) * 2048 )` For 32-bit platform on Windows: 100 through 524,288 buffers (`524,288 = 512 * 1024`) For 64-bit platforms: 100 through 2^{31}-1 buffers (For the actual value for your 64-bit platform, see your release notes. The maximum number of buffers on Solaris is 536,870,912.)
lrus	Specifies the number of LRU (least-recently-used) queues in the shared-memory buffer pool for the page size. You can tune the value of LRUS, in combination with the **lru_min_dirty** and **lru_max_dirty** values, to control how frequently the shared-memory buffers are flushed to disk. Setting LRUS too high might result in excessive page-cleaner activity.	`1` through `128`
lru_min_dirty	Specifies the percentage of modified pages in the LRU queues at which page cleaning is no longer mandatory. Page cleaners might continue cleaning beyond this point under some circumstances. The LRU values for flushing the bufferpool between checkpoints are not critical for checkpoint performance. The **lru_min_dirty** value is usually only necessary for maintaining enough clean pages for page replacement. Start by setting the LRU flushing parameters to set **lru_min_dirty** to `70`. For more information, see "LRU Values for Flushing a Buffer Pool Between Checkpoints" on page 15-6.	`0` through `100` (fractional values are allowed) If a parameter is specified out of the range of values, then the default of `50.00` percent is set.

Field	Explanation	Range of Values
lru_max_dirty	Specifies the percentage of modified pages in the LRU queues at which the queue is cleaned. The LRU values for flushing the bufferpool between checkpoints are not critical for checkpoint performance. The **lru_max_dirty** value is usually only necessary for maintaining enough clean pages for page replacement. Start by setting the LRU flushing parameters to set**lru_max_dirty** to 80. For more information, see "LRU Values for Flushing a Buffer Pool Between Checkpoints" on page 15-6.	0 through 100 (fractional values are allowed) If a parameter is specified out of the range of values, then the default of 60.00 percent is set.

If the database server is in online, quiescent, or administration mode, you can also use the **onparams** utility to add a new buffer pool of a different size. When you use the **onparams** utility, the information you specify is automatically transferred to the ONCONFIG file and new values are specified using the BUFFERPOOL keyword. You cannot change the values by editing the **onconfig.std** file.

When you use the **onparams** utility, you specify information as follows:

```
onparams -b -g <size of buffer page in Kbytes> -n <number of buffers>
-r <number of LRUs> -x <max dirty (fractional value allowed)>
-m <minimum dirty (fractional value allowed)>
```

For example:

```
onparams -b -g 6 -n 3000 -r 2 -x 2.0 -m 1.0
```

This adds 3000 buffers of size 6K bytes each with 2 LRUS and **lru_max_dirty** set at 2 percent and **lru_min_dirty** set at 1 percent.

For more information on the **onparams** utility, see the *IBM Informix Dynamic Server Administrator's Reference*.

Recommendation: Set the PHYSBUFF configuration parameter to at least 128 kilobytes. If the database server is configured to use RTO_SERVER_RESTART, set the PHYSBUFF configuration parameter to at least 512 kilobytes. Setting PHYSBUFF to a lower value could impact transaction performance and will result in a performance warning during server initialization.

The LG_ADDBPOOL log record and the **sysbufpool** system catalog table contain information on each buffer pool.

Buffer pools that are added while the database server is running go into virtual memory, not into resident memory. Only those buffer pool entries that are specified in the ONCONFIG file at startup go into resident memory, depending on the availability of the memory you are using.

Resizing an Existing Buffer Pool:

If you need to resize an existing buffer pool, you must bring down the database server. Then change the buffer pool size in the ONCONFIG file.

Deleting an Existing Buffer Pool:

If you need to delete an existing buffer pool, you must bring down the database server. Then, in the ONCONFIG file, delete the buffer pool.

Defining the Page Size

Use the **onspaces -k** option to set the page size in kilobytes, as follows:

```
onspaces -c -d DBspace [-t] [-k pagesize] -p path -o offset -s size [-m path offset]
```

The root dbspace is the default page size.

If you specify a page size, the page size must be a multiple of the default page size, but not greater than 16 kilobytes.

If you are using ON-Monitor to create a dbspace, be sure to enter the page size in kilobytes in the **Page Size** field.

Improving the Performance of Cooked-File Dbspaces by Using Direct I/O

On UNIX systems, you can improve the performance of cooked files used for dbspace chunks by using direct I/O.

Dynamic Server allows you to use either raw devices or cooked files for dbspace chunks. In general, cooked files are slower because of the additional overhead and buffering provided by the file system. Direct I/O bypasses the use of the file system buffers, and therefore is more efficient for reads and writes that go to disk. You specify direct I/O with the DIRECT_IO configuration parameter. If your file system supports direct I/O for the page size used for the dbspace chunk and you use direct I/O, performance for cooked files can approach the performance of raw devices used for dbspace chunks.

Prerequisites: Direct I/O must be available and the file system must support direct I/0 for the page size used for the dbspace chunk.

To improve the performance of cooked-file dbspaces by using direct I/O:

1. Verify that you have direct I/O and the file system supports direct I/O for the page size used for the dbspace chunk.
2. Enable direct I/O by setting the DIRECT_IO configuration parameter to 1.

If you have an AIX operating system, you can also enable concurrent I/O for Dynamic Server to use with direct IO when reading and writing to chunks that use cooked files.

For more information on using direct IO or concurrent IO, see the *IBM Informix Dynamic Server Performance Guide*.

Creating a Temporary Dbspace

To specify where to allocate the temporary files, create temporary dbspaces.

To define temporary dbspaces

1. Use the onspaces utility with the **-c -d -t** options.

   For more information, refer to "Creating a Dbspace that Uses the Default Page Size" on page 10-7.

2. Use the **DBSPACETEMP** environment variables or the DBSPACETEMP configuration parameter to specify the dbspaces that the database server can use for temporary storage.

   The DBSPACETEMP configuration parameter can contain dbspaces with a non-default page size. Although you can include dbspaces with different page sizes in the parameter list for DBSPACETEMP, the database server only uses dbspaces with the same page size as the first listed dbspace.

   For further information on DBSPACETEMP, refer to the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.
3. If you create more than one temporary dbspace, the dbspaces should reside on separate disks to optimize the I/O.

If you are creating a temporary dbspace, you must make the database server aware of the existence of the newly created temporary dbspace by setting the DBSPACETEMP configuration variable, the **DBSPACETEMP** environment variable, or both. The database server does not begin to use the temporary dbspace until you take both of the following steps:

- Set the DBSPACETEMP configuration parameter, the **DBSPACETEMP** environment variable, or both.
- Restart the database server.

The following example shows how to create 5-megabyte temporary dbspace named **temp_space** with an offset of 5000 kilobytes:

```
onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5120
```

For more information, see "Temporary Dbspaces" on page 9-10.

What to Do If You Run Out of Disk Space

When the initial chunk of the dbspace that you are creating is a cooked file on UNIX or an NTFS file on Windows, the database server verifies that the disk space is sufficient for the initial chunk. If the size of the chunk is greater than the available space on the disk, a message is displayed and no dbspace is created. However, the cooked file that the database server created for the initial chunk is not removed. Its size represents the space left on your file system before you created the dbspace. Remove this file to reclaim the space.

Adding a Chunk to a Dbspace or Blobspace

You add a *chunk* when a dbspace, blobspace, or sbspace is becoming full or it needs more disk space. Use **onspaces** or ISA to add a chunk. For information on using ISA to add a chunk, see the ISA online help.

Important: The newly added chunk (and its associated mirror, if one exists) is available immediately. If you are adding a chunk to a mirrored storage space, you must also add a mirror chunk.

To add a chunk using onspaces

1. On UNIX, you must be logged in as user **informix** or **root** to add a chunk.

   On Windows, users in the **Informix-Admin** group can add a chunk.
2. Ensure that the database server is in online, administration, or quiescent mode, or the cleanup phase of fast-recovery mode.

3. Allocate disk space for the chunk, as described in "Allocating Disk Space" on page 10-1.
4. To add a chunk, use the **-a** option of **onspaces**.

   If the storage space is mirrored, you must specify the path name of both a primary chunk and mirror chunk.

   If you specify an incorrect path name, offset, or size, the database server does not create the chunk and displays an error message. Also see "What to Do If You Run Out of Disk Space" on page 10-16.
5. After you create the chunk, you must perform a level-0 backup of the root dbspace and the dbspace, blobspace, or sbspace that contains the chunk.

The following example adds a 10-megabyte mirror chunk to **blobsp3**. An offset of 200 kilobytes for both the primary and mirror chunk is specified. If you are not adding a mirror chunk, you can omit the **-m** option.

```
onspaces -a blobsp3 -p /dev/raw_dev1 -o 200 -s 10240  -m /dev/raw_dev2 200
```

The next example adds a 5-megabyte chunk of raw disk space, at an offset of 5200 kilobytes, to dbspace **dbspc3**.

```
onspaces -a dbspc3 \\.\e: -o 5200 -s 5120
```

For reference information on adding a chunk to a dbspace with **onspaces**, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Adding a Chunk with ON-Monitor (UNIX)

To add a chunk to a dbspace, follow these instructions:

1. Choose **Add Chunk > Dbspaces** option.
2. Use RETURN or the arrow keys to select the blobspace or dbspace that will receive the new chunk and press CTRL-B or F3.
3. The next screen indicates whether the blobspace or dbspace is mirrored. If it is, enter Y in the **Mirror** field.
4. If the dbspace to which you are adding the chunk is a temporary dbspace, enter Y in the **Temp** field.
5. If you indicated that the dbspace or blobspace is mirrored, you must specify both a primary chunk and mirror chunk.

   Enter the complete path name for the new primary chunk in the **Full Pathname** field of the primary-chunk section.
6. Specify an offset in the **Offset** field.
7. Enter the size of the chunk, in kilobytes, in the **Size** field.
8. If you are mirroring this chunk, enter the complete path name, size, and optional offset in the mirror-chunk section of the screen.

Renaming Dbspaces

You can use the **onspaces** utility to rename a dbspace if you are user **informix** or have DBA privileges and the database server is in quiescent mode (and not any other mode).

To rename a dbspace use the following **onspaces** utility command:

onspaces -ren *old_dbspace_name*-n *new_dbspace_name*

You can rename standard dbspaces and all other spaces, including blobspaces, smart blobspaces, temporary spaces, and external spaces. However, you cannot rename any critical dbspace, such as a root dbspace or a dbspace that contains physical logs.

You can rename a dbspace and an sbspace:

- When Enterprise Replication is enabled
- On a primary database server when data replication is enabled

You cannot rename a dbspace and an sbspace on a secondary database server or when the secondary database server is part of the Enterprise Replication configuration

The rename dbspace operation only changes the dbspace name; it does not reorganize data.

The rename dbspace command updates the dbspace name in all places where that name is stored. This includes reserved pages on disk, system catalogs, the ONCONFIG configuration file, and in-memory data structures.

Warning: After renaming a dbspace, perform a level-0 archive of the renamed dbspace and the root dbspace. For information, see the *IBM Informix Backup and Restore Guide*.

Additional Actions You might Need to Take After Renaming a Dbspace

If you rename a dbspace, you must rewrite and recompile any stored procedure code that references the old dbspace name. For example, if you have a stored procedures that contains the ALTER FRAGMENT keywords and a reference to the dbspace name, you must rewrite and recompile that stored procedure.

If you rename dbspaces that are specified in the DATASKIP configuration parameter, you must manually update the DATASKIP configuration parameter after renaming the dbspace.

Managing Dbspace Partitions

For fragmented tables that use expression-based or round-robin distribution schemes, you can create multiple *partitions*, which are collections of pages for a table or index, within a single dbspace.

Storing multiple table or index fragments in a single dbspace improves query performance over storing each fragment in a different dbspace and simplifies management of dbspaces. You can also use dbspace partitions to circumvent the current limitation of 16 million pages in a single dbspace, if you subdivide the table or index into multiple partitions that are stored in the same dbspace.

Suppose you are creating a fragmented table using an expression-based distribution scheme in which each expression specifies the datasets that are placed in particular fragments. You might decide to separate the data in the table with data from one month in one dbspace and data from the next 11 months in 11 other dbspaces. However, if you want to use only one dbspace for all the yearly data, you can create partitions so the data for each month is stored in a separate partition in one dbspace.

If you create a fragmented table with partitions, each row in the **sysfragments** system catalog contains a partition name in the **Partition** column. If you create a fragmented table without partitions, the name of the dbspace appears in the **Partition** column. The **Flags** column in the **sysfragments** catalog tells you if the fragmentation scheme has partitions.

You can create tables and indexes with partitions, and you can create, drop, and alter partition fragments using the PARTITION keyword and the partition name.

To create a fragmented table with partitions, use SQL syntax as shown in the following example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
     PARTITION part1 (a >=0 AND a < 5) IN dbspace1,
     PARTITION part2 (a >=5 AND a < 10) IN dbspace1
          ...
       ;
```

If you created a table or index fragment containing partitions, you must use syntax containing the partition name when you use the ALTER FRAGMENT statement, as shown in the following examples:

```
ALTER FRAGMENT ON TABLE tb1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (a >=0 AND a < 5) IN dbspace1,
    PARTITION part_2 (a >=5 AND a < 10) IN dbspace1;

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (a >=0 AND a < 5) IN dbspace1,
    PARTITION part_2 (a >=5 AND a < 10) IN dbspace1;
```

You can use the PARTITION BY EXPRESSION clause instead of the FRAGMENT BY EXPRESSION clause in CREATE TABLE, CREATE INDEX, and ALTER FRAGMENT ON INDEX statements as shown in this example:

```
ALTER FRAGMENT ON INDEX idx1 INIT PARTITION BY EXPRESSION
           PARTITION part1   (a <= 10) IN idxdbspc1,
           PARTITION part2   (a <= 20) IN idxdbspc1,
           PARTITION part3  (a <= 30) IN idxdbspc1;
```

Use ALTER FRAGMENT syntax to change fragmented tables and indexes that do not have partitions into tables and indexes that have partitions. The syntax below shows how you might convert a fragmented table with multiple dbspaces into a fragmented table with partitions:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1,
     (c1=20) IN dbs2;
ALTER FRAGMENT ON TABLE t1 MODIFY dbs2 TO PARTITION part_3 (c1=20)
 IN dbs1
```

The syntax below shows how you might convert a fragmented index into an index that contains partitions:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
CREATE INDEX ind1 ON t1 (c1) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
     PARTITION part_1 (c1=10) IN dbs1, PARTITION part_2 (c1=20) IN dbs1,
     PARTITION part_3 (c1=30) IN dbs1,
```

See the *IBM Informix Dynamic Server Performance Guide* for more information on fragmentation, including fragmentation guidelines, procedures for fragmenting

indexes, procedures for creating attached and detached indexes with partitions, and examples of SQL statements used to create attached and detached indexes containing partitions.

See the *IBM Informix Guide to SQL: Syntax* for more syntax details, including information on partitions in GRANT FRAGMENT, REVOKE FRAGMENT statements and details for using the DROP, DETACH and MODIFY clauses of the ALTER FRAGMENT statement.

For information on monitoring partitions, see "Monitoring Storage Spaces" on page 10-7

Managing Blobspaces

This section discusses how to create a blobspace and determine the blobpage size. The database server stores TEXT and BYTE data in dbspaces or blobspaces, but blobspaces are more efficient. For information on adding a chunk, see " Adding a Chunk to a Dbspace or Blobspace" on page 10-16.

For information on monitoring blobspaces, see "Monitoring Storage Spaces" on page 10-7

Creating a Blobspace

You can use **onspaces**, ISA, or ON–Monitor to create a blobspace. Specify a blobspace name of up to 128 bytes. The name must be unique and must begin with a letter or underscore. You can use letters, digits, underscores, and **$** characters in the name.

Important: You can mirror the blobspace when you create it if mirroring is enabled for the database server. Mirroring takes effect immediately.

Before you create a blobspace

1. Allocate disk space for the blobspace, as described in "Allocating Disk Space" on page 10-1.
2. Determine what blobpage size is optimal for your environment.
   For instructions, see "Determining Blobpage Size" on page 10-21.

To create a blobspace using onspaces

1. To create a blobspace on UNIX, you must be logged in as user **informix** or **root**.
   To create a blobspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is in online, administration, or quiescent mode, or the cleanup phase of fast-recovery mode.
3. To add a blobspace, use the **onspaces -c -b** options.
   a. Specify an explicit path name for the blobspace. If the blobspace is mirrored, you must specify the path name and size of both the primary chunk and mirror chunk.
   b. Use the **-o** option to specify an offset for the blobspace.
   c. Use the **-s** option to specify the size of the blobspace chunk, in kilobytes.
   d. Use the **-g** option to specify the blobpage size in terms of the number of disk pages per blobpages.

See “Determining Blobpage Size.” For example, if your database server instance has a disk-page size of 2 kilobytes, and you want your blobpages to have a size of 10 kilobytes, enter 5 in this field.

If you specify an incorrect path name, offset, or size, the database server does not create the blobspace and displays an error message. Also see “What to Do If You Run Out of Disk Space” on page 10-16.

4. After you create the blobspace, you must perform a level-0 backup of the root dbspace and the new blobspace.

The following shows how to create a 10-megabyte mirrored blobspace, **blobsp3**, with a blobpage size of 10 kilobytes, where the database server page size is 2 kilobytes. An offset of 200 kilobytes for the primary and mirror chunks is specified. The blobspace is created from raw disk space on UNIX.

```
onspaces -c -b blobsp3 -g 5 -p /dev/raw_dev1 -o 200 -s 10240 -m /dev/raw_dev2 200
```

For reference information on creating a blobspace with onspaces, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

To create a blobspace with ON-Monitor (UNIX)

1. Select the **Dbspaces > BLOBSpace** option.
2. Enter the name of the new blobspace in the **BLOBSpace Name** field.
3. If you want to create a mirror for the initial blobspace chunk, enter Y in the **Mirror** field.

   Otherwise, enter N.
4. Specify the blobpage size in terms of the number of disk pages per blobpage in the **BLOBPage Size** field.

   See “Determining Database Server Page Size” on page 10-22. For example, if your database server instance has a disk-page size of 2 kilobytes, and you want your blobpages to have a size of 10 kilobytes, enter 5 in this field.
5. Enter the complete path name for the initial primary chunk of the blobspace in the **Full Pathname** field of the primary-chunk section.
6. Specify an offset in the **Offset** field.
7. Enter the size of the chunk, in kilobytes, in the **Size** field.
8. If you are mirroring this blobspace, enter the full path name, size, and optional offset in the mirror-chunk section of the screen.

Preparing Blobspaces to Store TEXT and BYTE Data

A newly created blobspace is not immediately available for storage of TEXT or BYTE data. Blobspace logging and recovery require that the statement that creates a blobspace and the statements that insert TEXT and BYTE data into that blobspace appear in separate logical-log files. This requirement is true for all blobspaces, regardless of the logging status of the database. To accommodate this requirement, switch to the next logical-log file after you create a blobspace. (For instructions, see “Backing Up Log Files to Free Blobpages” on page 13-6.)

Determining Blobpage Size

When you create a blobspace, use the size of the most frequently occurring simple large object as the size of the blobpage. In other words, choose a blobpage size that wastes the least amount of space. For information on calculating an optimal

blobpage size, see blobpage size considerations in the chapter on effect of configuration on I/O activity in the *IBM Informix Dynamic Server Performance Guide*.

If a table has more than one TEXT or BYTE column, and the objects are not close in size, store each column in a different blobspace, each with an appropriately sized blobpage. See "Tables" on page 9-21.

Determining Database Server Page Size

When you specify blobpage size, you specify it in terms of the database server pages. You can use one of the following methods to determine the database server page size for your system:

- Run the **onstat -b** utility to display the system page size, given as buffer size on the last line of the output.
- To view the contents of the PAGE_PZERO reserved page, run the **oncheck -pr** utility.
- **UNIX only**: In ON–Monitor, select either the **Parameters > Shared-Memory** or **Parameters > Initialize** option to display the system page size.

Obtaining Blobspace Storage Statistics

To help you determine the optimal blobpage size for each blobspace, use the following database server utility commands:

- **oncheck -pe**
- **oncheck -pB**

The **oncheck -pe** command provides background information about the objects stored in a blobspace:

- Complete ownership information (displayed as *database:owner.table*) for each table that has data stored in the blobspace chunk
- The total number of pages used by each table to store its associated TEXT and BYTE data
- The total free and total overhead pages in the blobspace

The **oncheck -pB** command lists the following statistics for each table or database:

- The number of blobpages used by the table or database in each blobspace
- The average fullness of the blobpages used by each simple large object stored as part of the table or database

For more information, see "Monitoring Blobspace Usage with oncheck -pe" on page 10-40, "Determining Blobpage Fullness with oncheck -pB" on page 10-39, and optimizing blobspace blobpage size in the chapter on table performance considerations in the *IBM Informix Dynamic Server Performance Guide*.

Managing Sbspaces

This section describes how to create a standard or temporary sbspace, monitor the metadata and user-data areas, add a chunk to an sbspace, and alter storage characteristics of smart large objects.

For information on monitoring sbspaces, see "Monitoring Storage Spaces" on page 10-7

Creating an Sbspace

Use the onspaces utility or ISA to create an sbspace.

To create an sbspace using onspaces

1. To create an sbspace on UNIX, you must be logged in as user **informix** or **root**.

   To create an sbspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is online, administration, or quiescent mode, or in the cleanup phase of fast-recovery mode.
3. Use the **onspaces -c -S** options to create the sbspace.
   a. Use the **-p** option to specify the path name, the **-o** option to specify the offset, and the **-s** option to specify the sbspace size.
   b. If you wish to mirror the sbspace, use the **-m** option to specify the mirror path and offset.
   c. If you want to use the default storage characteristics for the sbspace, omit the **-Df** option.

      If you want to specify different storage characteristics, use the **-Df** option. For more information, refer to "Storage Characteristics of Sbspaces" on page 9-14.
   d. The first chunk in an sbspace must have a metadata area.

      You can specify a metadata area for an sbspace or let the database server calculate the size of the metadata area. For more information, see " Sizing Sbspace Metadata" on page 10-24.
4. After you create the sbspace, you must perform a level-0 backup of the root dbspace and the new sbspace.
5. To start storing smart large objects in this sbspace, specify the space name in the SBSPACENAME configuration parameter.
6. Use **onstat -d**, **onstat -g smb s**, and **oncheck -cs**, -**cS**, **-ps**, or **-pS** to display information about the sbspace.

   For more information, see "Monitoring Sbspaces" on page 10-41.

This shows how to create a 20-megabyte mirrored sbspace, **sbsp4**. Offsets of 500 kilobytes for the primary and 500 kilobytes for the mirror chunks are specified, as well as a metadata size of 150 kilobytes with a 200 kilobyte offset. The AVG_LO_SIZE **-Df** tag specifies an expected average smart-large-object size of 32 kilobytes.

```
onspaces -c -S sbsp4 -p /dev/rawdev1 -o 500 -s 20480 -m /dev/rawdev2 500
-Ms 150 -Mo 200 -Df "AVG_LO_SIZE=32"
```

For information about creating an sbspace and default options for smart large objects, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*. For information on creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

To create an sbspace using ISA

1. Create the sbspace using ISA. For more information, see the ISA online help.
2. Back up the new sbspace and the root dbspace.

Sizing Sbspace Metadata

The first chunk of an sbspace must have a metadata area. When you add smart large objects and chunks to the sbspace, the metadata area grows. In addition, the database server reserves 40 percent of the user area to be used in case the metadata area runs out of space.

It is important to size the metadata area for an sbspace correctly to ensure that the sbspace does not run out of metadata space. You can either:

- Let the database server calculate the size of the metadata area for the new sbspace chunk.
- Specify the size of the metadata area explicitly.

For instructions on estimating the size of the sbspace and metadata area, see table performance considerations in the *IBM Informix Dynamic Server Performance Guide*. Also see "Monitoring the Metadata and User-Data Areas" on page 10-45.

Adding a Chunk to an Sbspace

Use the **onspaces** utility or ISA to add a chunk to an sbspace or temporary sbspace. You can specify a metadata area for a chunk, let the database server calculate the metadata area, or use the chunk for user data only.

To add a chunk to an sbspace using onspaces

1. Ensure that the database server is online, administration, or quiescent mode, or in the cleanup phase of fast-recovery mode.
2. Use the **onspaces -a** option to create the sbspace chunk.
   a. Use the **-p** option to specify the path name, the **-o** option to specify the offset, and the **-s** option to specify the chunk size.
   b. If you want to mirror the chunk, use the **-m** option to specify the mirror path and offset.
   c. To specify the size and offset of the metadata space, use the **-Mo** and **-Ms** options.

      The database server allocates the specified amount of metadata area on the new chunk.
   d. To allow the database server to calculate the size of the metadata for the new chunk, omit the **-Mo** and **-Ms** options.

      The database server divides the estimated average size of the smart large objects by the size of the user data area.
   e. To use the chunk for user data only, specify the **-U** option.

      If you use the **-U** option, the database server does not allocate metadata space in this chunk. Instead, the sbspace uses the metadata area in one of the other chunks.
3. After you add a chunk to the sbspace, the database server writes the CHRESERV and CHKADJUP log records.
4. Perform a level-0 backup of the root dbspace and the sbspace.
5. Use **onstat -d** and **oncheck -pe** to monitor the amount of free space in the sbspace chunk.

   For more information, see "Monitoring Sbspaces" on page 10-41.

This example adds a 10-megabyte mirror chunk to **sbsp4**. An offset of 200 kilobytes for both the primary and mirror chunk is specified. If you are not adding a mirror chunk, you can omit the **-m** option. The **-U** option specifies that the new chunk will contain user data exclusively.

```
onspaces -a sbsp4 -p /dev/rawdev1 -o 200 -s 10240 -m /dev/rawdev2 200 -U
```

For more information, see " Adding a Chunk to a Dbspace or Blobspace" on page 10-16, and information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Altering Storage Characteristics of Smart Large Objects

Use the **onspaces -ch** command to change the following default storage characteristics for the sbspace:

- Extent sizes
- Average smart-large-object size
- Buffering mode
- Last-access time
- Lock mode
- Logging

For more information, refer to "Storage Characteristics of Sbspaces" on page 9-14 and managing sbspaces in the chapter on table performance considerations in your *IBM Informix Dynamic Server Performance Guide*.

Creating a Temporary Sbspace

For background information and the rules for determining where temporary smart large objects are stored, see "Temporary Sbspaces" on page 9-18. You can store temporary smart large objects in a standard or temporary sbspace. You can add or drop chunks in a temporary sbspace.

To create a temporary sbspace with a temporary smart large object

1. Allocate space for the temporary sbspace. For details, see "Allocating Disk Space" on page 10-1.

   For information on SBSPACETEMP, see the configuration parameters chapter in the *IBM Informix Administrator's Reference*.
2. Create the temporary sbspace as the following example shows:

   ```
   onspaces -c -S tempsbsp -t -p ./tempsbsp -o 0 -s 1000
   ```

   You can specify any of the following **onspaces** options:

   a. Specify a metadata area and offset (**-Ms** and **-Mo**).

   b. Specify storage characteristics (**-Df**).

      You cannot turn on logging for a temporary sbspace.
3. Set the SBSPACETEMP configuration parameter to the name of the default temporary sbspace storage area.

   Restart the database server.
4. Use **onstat -d** to display the temporary sbspace.

   For an information on and an example of **onstat -d** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

5. Specify the LO_CREATE_TEMP flag when you create a temporary smart large object.

   Using DataBlade API:

   ```
   mi_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
   ```

   Using Informix ESQL/C:

   ```
   ifx_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
   ```

For information on creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Dropping a Chunk

Use **onspaces** or ISA to drop a chunk from a dbspace.

Before you drop a chunk, ensure that the database server is in the correct mode, using the following table as a guideline.

Chunk Type	Database Server in Online Mode	Database Server in Administration or Quiescent Mode	Database Server in Offline Mode
Dbspace chunk	Yes	Yes	No
Temporary dbspace chunk	Yes	Yes	No
Blobspace chunk	No	Yes	No
Sbspace or temporary sbspace chunk	Yes	Yes	No

Verifying Whether a Chunk Is Empty

To drop a chunk successfully from a dbspace with either of these utilities, the chunk must not contain any data. All pages other than overhead pages must be freed.

If any pages remain allocated to nonoverhead entities, the utility returns the following error:

```
Chunk is not empty.
```

In addition, when a dbspace consists of two or more chunks and the additional chunks do not contain user data, the additional chunks cannot be deleted if the chunks contain a tblspace **tblspace**.

If you receive the "`Chunk is not empty`" message, you must determine which table or other entity still occupies space in the chunk by executing **oncheck -pe** to list contents of the extent.

Usually, the pages can be removed when you drop the table that owns them. Then reenter the utility command.

Dropping a Chunk from a Dbspace with onspaces

The following example drops a chunk from **dbsp3** on UNIX. An offset of 300 kilobytes is specified.

```
onspaces -d dbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of a dbspace with the syntax in the previous example. Instead, you must drop the dbspace. Use the **fchunk** column of **onstat -d** to determine which is the initial chunk of a dbspace. For more information about **onstat**, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

For information about dropping a chunk from a dbspace with **onspaces**, see the *IBM Informix Administrator's Reference*.

Dropping a Chunk from a Blobspace

The procedure for dropping a chunk from a blobspace is identical to the procedure for dropping a chunk from a dbspace described in "Dropping a Chunk from a Dbspace with onspaces" on page 10-26 except that the database server must be in quiescent or administration mode. Other than this condition, you need only substitute the name of your blobspace wherever a reference to a dbspace occurs.

Dropping a Chunk from an Sbspace with onspaces

The following example drops a chunk from **sbsp3** on UNIX. An offset of 300 kilobytes is specified. The database server must be in online administration, or quiescent mode when you drop a chunk from an sbspace or temporary sbspace.

```
onspaces -d sbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of an sbspace with the syntax in the previous example. Instead, you must drop the sbspace. Use the **fchunk** column of **onstat -d** to determine which chunk is the initial chunk of an sbspace.

Using the -f (Force) Option

You can use the **-f** option of **onspaces** to drop an sbspace chunk without metadata allocated in it. If the chunk contains metadata for the sbspace, you must drop the entire sbspace. Use the **Chunks** section of **onstat -d** to determine which sbspace chunks contain metadata.

```
onspaces -d sbsp3 -f
```

Warning: If you force the drop of an sbspace, you might introduce consistency problems between tables and sbspaces.

Deleting Smart Large Objects Without any Pointers

Each smart large object has a reference count, the number of pointers to the smart large object. When the reference count is greater than 0, the database server assumes that the smart large object is in use and does not delete it.

Rarely, a smart large object with a reference count of 0 remains. You can use the **onspaces -cl** command to delete all smart large objects that have a reference count of 0, if it is not open by any application.

For information on using **onspaces -cl**, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Dropping a Storage Space

Use **onspaces**, ISA, or ON–Monitor to drop a dbspace, temporary dbspace, blobspace, sbspace, temporary sbspace, or extspace.

On UNIX, you must be logged in as root or **informix** to drop a storage space. On Windows, you must be a member of the **Informix-Admin** group to drop a storage space.

You can drop a storage space only when the database server is in online, administration, or quiescent mode.

Preparing to Drop a Storage Space

Before you drop a dbspace, you must first drop all databases and tables that you previously created in that dbspace. You cannot drop the root dbspace.

Before you drop a blobspace, you must drop all tables that have a TEXT or BYTE column that references the blobspace.

Execute **oncheck -pe** to verify that no tables or log files reside in the dbspace or blobspace.

Before you drop an sbspace, you must drop all tables that have a CLOB or BLOB column that reference objects that are stored in the sbspace. For sbspaces, you need not delete columns that point to an sbspace, but these columns must be null; that is, all smart large objects must be deallocated from the sbspace.

Tip: If you drop tables on dbspaces where light appends are occurring, the light appends might be slower than you expect. The symptom of this problem is physical logging activity. If light appends are slower than you expect, make sure that no tables are dropped in the dbspace either before or during the light appends. If you have dropped tables, force a checkpoint with **onmode -c** before you perform the light append.

Dropping a Mirrored Storage Space

If you drop a storage space that is mirrored, the mirror spaces are also dropped.

If you want to drop only a storage-space mirror, turn off mirroring. (See "Ending Mirroring" on page 18-6.) This action drops the dbspace, blobspace, or sbspace mirrors and frees the chunks for other uses.

Dropping a Storage Space with onspaces

To drop a storage space with onspaces, use the **-d** option as illustrated in the following examples.

This example drops a dbspace called **dbspce5** and its mirrors.

```
onspaces -d dbspce5
```

This example drops a dbspace called **blobsp3** and its mirrors.

```
onspaces -d blobsp3
```

Use the **-d** option with the **-f** option if you want to drop an sbspace that contains data. If you omit the **-f** option, you cannot drop an sbspace that contains data. This example drops an sbspace called **sbspc4** and its mirrors.

```
onspaces -d sbspc4 -f
```

Warning: If you use the **-f** option, the tables in the database server might have dead pointers to the deleted smart large objects.

For information on dropping a storage space with **onspaces**, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Dropping a Dbspace or Blobspace with ON-Monitor (UNIX)

To drop a dbspace or blobspace with ON–Monitor, follow these instructions:

1. Select the **Dbspaces > Drop** option.
2. Use `RETURN` or arrow keys to scroll to the dbspace or blobspace that you want to drop.
3. Press `CTRL-B` or `F3`.

You are asked to confirm that you want to drop the dbspace or blobspace.

Backing Up After Dropping a Storage Space

If you create a storage space with the same name as the deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see the *IBM Informix Backup and Restore Guide*.

Warning: After you drop a dbspace, blobspace, or sbspace, the newly freed chunks are available for reassignment to other dbspaces, blobspaces, or sbspaces. However, before you reassign the newly freed chunks, you must perform a level-0 backup of the root dbspace and the modified storage space. If you do not perform this backup, and you subsequently need to perform a restore, the restore might fail because the backup reserved pages are not up-to-date.

Managing Extspaces

An extspace does not require allocation of disk space. You create and drop extspaces using the **onspaces** utility. For more information about extspaces, refer to "Extspaces" on page 9-20.

Creating an Extspace

You create an extspace with the **onspaces** utility. But you should first have a valid data source and a valid access method with which to access that data source. Although you can create an extspace without a valid access method or a valid data source, any attempts to retrieve data from the extspace will generate an error. For information on access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*.

To create an extspace with **onspaces**, use the -c option as illustrated in the following example. The following example shows how to create an extspace, **pass_space**, that is associated with the UNIX password file.

```
onspaces -c -x pass_space -l /etc/passwd
```

Specify an extspace name of up to 128 bytes. The name must be unique and begin with a letter or underscore. You can use letters, digits, underscores, and **$** characters in the name.

Important: The preceding example assumes that you have coded a routine that provides functions for correctly accessing the file **passwd** and that the file itself exists. After you have created the extspace, you must use the appropriate commands to allow access to the data in the file **passwd**. For more information on user-defined access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*.

For reference information on creating an extspace with **onspaces**, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Dropping an Extspace

To drop an extspace with onspaces, use the **-d** option as illustrated in the following examples. An extspace cannot be dropped if it is associated with an existing table or index.

This example drops an extspace called **pass_space**.

```
onspaces -d pass_space
```

Skipping Inaccessible Fragments

One benefit that fragmentation provides is the ability to skip table fragments that are unavailable during an I/O operation. For example, a query can proceed even when a fragment is located on a chunk that is currently down as a result of a disk failure. When this situation occurs, a disk failure affects only a portion of the data in the fragmented table. By contrast, tables that are not fragmented can become completely inaccessible if they are located on a disk that fails.

This functionality is controlled as follows:

- By the database server administrator with the DATASKIP configuration parameter
- By individual applications with the SET DATASKIP statement

Using the DATASKIP Configuration Parameter

You can set the DATASKIP parameter to OFF, ALL, or ON *dbspace_list.* OFF means that the database server does not skip any fragments. If a fragment is unavailable, the query returns an error. ALL indicates that any unavailable fragment is skipped. ON *dbspace_list* instructs the database server to skip any fragments that are located in the specified dbspaces.

Using the Dataskip Feature of onspaces

Use the dataskip feature of the onspaces utility to specify the dbspaces that are to be skipped when they are unavailable. For example, the following command sets the DATASKIP parameter so that the database server skips the fragments in **dbspace1** and **dbspace3**, but not in **dbspace2**:

```
onspaces -f ON dbspace1 dbspace3
```

For the complete syntax of this **onspaces** option, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Using onstat to Check Dataskip Status

Use the **onstat** utility to list the dbspaces currently affected by the dataskip feature. The **-f** option lists both the dbspaces that were set with the DATASKIP configuration parameter and the **-f** option of the **onspaces** utility.

When you execute **onstat -f**, you receive a message that tells you whether the DATASKIP configuration parameter is set to **on** for all dbspaces, **off** for all dbspaces, or **on** for specific dbspaces.

Using the SQL Statement SET DATASKIP

An application can use the SQL statement SET DATASKIP to control whether a fragment should be skipped if it is unavailable. Applications should include this statement only in limited circumstances, because it causes queries to return different results, depending on the availability of the underlying fragments. Like the configuration parameter DATASKIP, the SET DATASKIP statement accepts a list of dbspaces that indicate to the database server which fragments to skip. For example, suppose that an application programmer included the following statement at the beginning of an application:

```
SET DATASKIP ON dbspace1, dbspace5
```

This statement causes the database server to skip **dbspace1** or **dbspace5** whenever both of these conditions are met:

- The application attempts to access one of the dbspaces.
- The database server finds that one of the dbspaces is unavailable.

If the database server finds that both **dbspace1** and **dbspace5** are unavailable, it skips both dbspaces.

The DEFAULT setting for the SET DATASKIP statement allows a database server administrator to control the dataskip feature. Suppose that an application developer includes the following statement in an application:

```
SET DATASKIP DEFAULT
```

When a query is executed subsequent to this SQL statement, the database server checks the value of the configuration parameter DATASKIP. Encouraging end users to use this setting allows the database server administrator to specify which dbspaces are to be skipped as soon as the database server administrator becomes aware that one or more dbspaces are unavailable.

Effect of the Dataskip Feature on Transactions

If you turn the dataskip feature on, a SELECT statement always executes. In addition, an INSERT statement always succeeds if the table is fragmented by round-robin and at least one fragment is online. However, the database server *does not* complete operations that write to the database if a possibility exists that such operations might compromise the integrity of the database. The following operations fail:

- All UPDATE and DELETE operations where the database server cannot eliminate the down fragments

If the database server *can* eliminate the down fragments, the update or delete is successful, but this outcome is independent of the DATASKIP setting.

- An INSERT operation for a table fragmented according to an expression-based distribution scheme where the appropriate fragment is down
- Any operation that involves referential constraint checking if the constraint involves data in a down fragment

  For example, if an application deletes a row that has child rows, the child rows must also be available for deletion.
- Any operation that affects an index value (for example, updates to a column that is indexed) where the index in question is located in a down chunk

Determining When to Use Dataskip

Use this feature sparingly and with caution because the results are always suspect. Consider using it in the following situations:

- You can accept the compromised integrity of transactions.
- You can determine that the integrity of the transaction is not compromised.

The latter task can be difficult and time consuming.

Determining When to Skip Selected Fragments

In certain circumstances, you might want the database server to skip some fragments, but not others. This usually occurs in the following situations:

- Fragments can be skipped because they do not contribute significantly to a query result.
- Certain fragments are down, and you decide that skipping these fragments and returning a limited amount of data is preferable to canceling a query.

When you want to skip fragments, use the ON *dbspace-list* setting to specify a list of dbspaces with the fragments that the database server should skip.

Determining When to Skip All Fragments

Setting the DATASKIP configuration parameter to ALL causes the database server to skip all unavailable fragments. Use this option with caution. If a dbspace becomes unavailable, all queries initiated by applications that do not issue a SET DATASKIP OFF statement before they execute could be subject to errors.

Monitoring Fragmentation Use

The database administrator might find the following aspects of fragmentation useful to monitor:

- Data distribution over fragments
- I/O request balancing over fragments
- The status of chunks that contain fragments

The administrator can monitor the distribution of data over table fragments. If the goal of fragmentation is improved administration response time, it is important for data to be distributed evenly over the fragments. To monitor fragmentation disk use, you must monitor database server tblspaces, because the unit of disk storage

for a fragment is a tblspace. (For information on how to monitor the data distribution for a fragmented table, see "Monitoring Tblspaces and Extents" on page 10-38.)

The administrator must monitor I/O request queues for data that is contained in fragments. When I/O queues become unbalanced, the administrator should work with the DBA to tune the fragmentation strategy. (For a discussion of how to monitor chunk use, including the I/O queues for each chunk, see "Monitoring Chunks" on page 10-34.)

The administrator must monitor fragments for availability and take appropriate steps when a dbspace that contains one or more fragments fails. For how to determine if a chunk is down, see "Monitoring Chunks" on page 10-34.

Displaying Databases

You can display databases that you create with the following tools:

- SMI tables
- ISA
- ON–Monitor

Using SMI Tables

Query the **sysdatabases** table to display a row for each database managed by the database server. For a description of the columns in this table, see the **sysdatabases** information in the chapter about the **sysmaster** database in the *IBM Informix Administrator's Reference*.

Using ISA

To query **sysdatabases** using ISA, follow these steps:

1. Choose **SQL > Query**.
2. Select the **sysmaster** data in the **Database** list.
3. Enter the following command and click **Submit**:

```
select * from sysdatabases;
```

Using ON-Monitor (UNIX)

To use ON–Monitor to find the current status of each database, select the **Status > Databases** option. ON–Monitor can only display up to 100 databases. If you have more than 100 databases on your database server, use the SMI tables to display the full list, as described in the previous section.

Monitoring Disk Usage

This section describes methods of tracking the disk space used by various database server storage units.

For background information about internal database server storage units mentioned in this section, see the chapter about disk structures and storage in the *IBM Informix Administrator's Reference*.

Monitoring Chunks

You can monitor chunks for the following information:

- Chunk size
- Number of free pages
- Tables within the chunk

This information allows you to track the disk space used by chunks, monitor chunk I/O activity, and check for fragmentation.

onstat -d

The **onstat -d** utility lists all dbspaces, blobspaces, and sbspaces and the following information for the chunks within those spaces.

- The address of the chunk
- The chunk number and associated dbspace number
- The offset into the device (in pages)
- The size of the chunk (in pages)
- The number of free pages in the chunk
- The path name of the physical device

If you issue the **onstat -d** command on an instance with blobspace chunks, the number of free pages shown is out of date. The tilde (~) that precedes the `free` value indicates that this number is approximate. The **onstat -d** command does not register a blobpage as available until the logical log in which a deletion occurred is backed up and the blobpage is freed. Therefore, if you delete 25 simple large objects and immediately execute **onstat -d**, the newly freed space does not appear in the **onstat** output.

To obtain an accurate number of free blobpages in a blobspace chunk, issue the **onstat -d update** command. For details, see "onstat -d update."

In **onstat -d update** output, the **flags** column in the **chunk** section provides the following information:

- Whether the chunk is the primary chunk or the mirror chunk
- Whether the chunk is online, is down, is being recovered, or is a new chunk

For an example of **onstat -d** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

Important: You must perform a level-0 backup of the root dbspace and the modified dbspace before mirroring can become active and after turning off mirroring.

onstat -d update

The **onstat -d update** option displays the same information as **onstat -d** and an accurate number of free blobpages for each blobspace chunk.

onstat -D

The **onstat -D** option displays the same information as **onstat -d**, plus the number of pages read from the chunk (in the **page Rd** field).

onstat -g iof

The **onstat -g iof** option displays the number of reads from each chunk and the number of writes to each chunk. If one chunk has a disproportionate amount of I/O activity against it, this chunk might be a system bottleneck. This option is useful for monitoring the distribution of I/O requests against the different fragments of a fragmented table.

For an example of **onstat -g iof** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

oncheck -pr

The database server stores chunk information in the reserved pages PAGE_1PCHUNK and PAGE_2PCHUNK.

To list the contents of the reserve pages, execute **oncheck -pr**. Figure 10-1 shows sample output for **oncheck -pr**. This output is essentially the same as the **onstat -d** output; however, if the chunk information has changed since the last checkpoint, these changes do not appear in the **oncheck -pr** output.

```
Validating PAGE_1DBSP & PAGE_2DBSP...
          Using dbspace page PAGE_2DBSP.

    DBspace number                1
    DBspace name                  rootdbs
    Flags                         0x20001          No mirror chunks
    Number of chunks              2
    First chunk                   1
    Date/Time created             07/28/2008 14:46:55
    Partition table page number   14
    Logical Log Unique Id         0
    Logical Log Position          0
    Oldest Logical Log Unique Id  0
    Last Logical Log Unique Id    0
    Dbspace archive status        No archives have occurred
.
.
Validating PAGE_1PCHUNK & PAGE_2PCHUNK...
          Using primary chunk page PAGE_2PCHUNK.

    Chunk number                  1
    Flags                         0x40       Chunk is online
    Chunk path                    /home/server/root_chunk
    Chunk offset                  0 (p)
    Chunk size                    75000 (p)
    Number of free pages          40502
    DBSpace number                1
.
.
.
```

Figure 10-1. oncheck -pr Output Showing Dbspace and Chunk Information

oncheck -pe

To obtain the physical layout of information in the chunk, execute **oncheck -pe**. The dbspaces, blobspaces, and sbspaces are listed. Figure 10-2 on page 10-36 shows sample output for **oncheck -pe**.

The following information is displayed:

- The name, owner, and creation date of the dbspace
- The size in pages of the chunk, the number of pages used, and the number of pages free
- A listing of all the tables in the chunk, with the initial page number and the length of the table in pages

The tables within a chunk are listed sequentially. This output is useful for determining chunk fragmentation. If the database server is unable to allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly fragmented.

```
DBSpace Usage Report:  rootdbs          Owner:  informix  Created: 08/08/2006

Chunk  Pathname                                  Size     Used     Free
    1  /home/server/root_chunk                  75000    19420    55580

Description                                             Offset    Size
----------------------------------------         --------------------
RESERVED PAGES                                             0        12
CHUNK FREELIST PAGE                                       12         1
rootdbs:'informix'.TBLSpace                               13       250
PHYSICAL LOG                                             263      1000
FREE                                                    1263      1500
LOGICAL LOG: Log file 2                                 2763      1500
LOGICAL LOG: Log file 3                                 4263      1500
...
sysmaster:'informix'.sysdatabases                      10263         4
sysmaster:'informix'.systables                         10267         8
...

Chunk  Pathname                                 Size     Used     Free
    2  /home/server/dbspace1                    5000       53     4947

Description                                             Offset    Size
----------------------------------------         --------------------
RESERVED PAGES                                             0         2
CHUNK FREELIST PAGE                                        2         1
dbspace1:'informix'.TBLSpace                               3        50
FREE                                                      53      4947
```

Figure 10-2. oncheck -pe Output

Using IBM Informix Server Administrator

You can perform the following tasks using ISA commands:

- Check reserved pages.
- Check storage spaces.
- Add dbspaces, temporary dbspaces, blobspaces, temporary sbspaces, and sbspaces.
- Display and add chunks to a storage space.
- Check the dataskip status.
- Display and add external spaces.
- Display the number of pages in your database, percentage of allocated space, and used space.
- Override ONDBSPACEDOWN.

Using ON-Monitor (UNIX)

You can perform the following tasks using ON–Monitor commands.

ON_Monitor Command
Description

Status > Spaces
Displays status information about storage spaces and chunks

Dbspaces > Create
Creates a dbspace

Dbspaces > BLOBSpace
Creates a blobspace

Dbspaces > Mirror
Adds or drops mirroring for a storage space

Dbspaces > Info
Displays information about storage spaces

Dbspaces > Add Chunk
Adds a chunk to a storage space

Dbspaces > dataSkip
Starts or stops dataskip

Dbspaces > Chunk
Adds a chunk to a dbspace or blobspace

Dbspaces > Drop
Drops a dbspace or blobspace

Dbspaces > Status
Changes the mirror status of a chunk

Using SMI Tables

Query the **syschunks** table to obtain the status of a chunk. The following columns are relevant.

Column
Description

chknum
Number of the chunk within the dbspace

dbsnum
Number of the dbspace

chksize
Total size of the chunk in pages

nfree Number of pages that are free

is_offline
Whether the chunk is down

is_recovering
Whether the chunk is recovering

mis_offline
Whether the mirror chunk is down

mis_recovering
Whether the mirror chunk is being recovered

The **syschkio** table contains the following columns.

Column	Description
pagesread	Number of pages read from the chunk
pageswritten	Number of pages written to the chunk

Monitoring Tblspaces and Extents

Monitor tblspaces and extents to determine disk usage by database, table, or table fragment. Monitoring disk usage by table is particularly important when you are using table fragmentation, and you want to ensure that table data and table index data are distributed appropriately over the fragments.

Execute **oncheck -pt** to obtain extent information. The **oncheck -pT** option returns all the information from the **oncheck -pt** option as well as the additional information about page and index usage.

Using SMI Tables

Query the **systabnames** table to obtain information about each tblspace. The **systabnames** table has columns that indicate the corresponding table, database, and table owner for each tblspace.

Query the **sysextents** table to obtain information about each extent. The **sysextents** table has columns that indicate the database and the table that the extent belongs to, as well as the physical address and size of the extent.

Monitoring Simple Large Objects in a Blobspace

Monitor blobspaces to determine the available space and whether the blobpage size is optimal.

onstat -O

The **onstat -O** option displays information about the staging-area blobspace and the Optical Subsystem memory cache. The totals shown in the display accumulate from session to session. The database server resets the totals to `0` only when you execute **onstat -z**.

For an example of **onstat -O** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

The first section of the display describes the following system-cache totals information.

Column	Description
size	Size specified in the OPCACHEMAX configuration parameter
alloc	Number of 1-kilobyte pieces that the database server allocated to the cache
avail	Portion of alloc (in kilobytes) not used

number
Number of simple large objects that the database server successfully put into the cache without overflowing

kbytes
Number of kilobytes of the simple large objects that the database server put into the cache without overflowing

number
Number of simple large objects that the database server wrote to the staging-area blobspace

kbytes
Number of kilobytes of simple large objects that the database server wrote to the staging-area blobspace

Although the **size** output indicates the amount of memory that is specified in the configuration parameter OPCACHEMAX, the database server does not allocate memory to OPCACHEMAX until necessary. Therefore, the **alloc** output reflects only the number of 1-kilobyte pieces of the largest simple large object that has been processed. When the values in the **alloc** and **avail** output are equal, the cache is empty.

The second section of the display describes the following user-cache totals information.

Column	Description
SID	Session ID for the user
user	User ID of the client
size	Size specified in the **INFORMIXOPCACHE** environment variable, if set If you do not set the **INFORMIXOPCACHE** environment variable, the database server uses the size that you specify in the configuration parameter OPCACHEMAX.
number	Number of simple large objects that the database server put into cache without overflowing
kbytes	Number of kilobytes of simple large objects that the database server put into the cache without overflowing
number	Number of simple large objects that the database server wrote to the staging-area blobspace
kbytes	Size of the simple large objects (in kilobytes) that the database server wrote to the staging-area blobspace

Determining Blobpage Fullness with oncheck -pB

The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobspace.

Execute **oncheck -pB** with either a database name or a table name as a parameter. The following example retrieves storage information for all simple large objects stored in the table **sriram.catalog** in the **stores_demo** database:

```
oncheck -pB stores_demo:sriram.catalog
```

For detailed information about interpreting the **oncheck -pB** output, see optimizing blobspace blobpage size in the chapter on table performance considerations in the *IBM Informix Dynamic Server Performance Guide*.

Monitoring Blobspace Usage with oncheck -pe

The **oncheck -pe** command provides information about blobspace usage:

- Names of the tables that store TEXT and BYTE data, by chunk
- Number of disk pages (*not* blobpages) used, by table
- Number of free disk pages remaining, by chunk
- Number of overhead pages used, by chunk

Figure 10-3 shows sample **oncheck -pe** output.

```
BLOBSpace Usage Report:  fstblob          Owner:  informix  Created: 03/01/08
     Chunk: 3   /home/server/blob_chunk                Size     Used     Free
                                                      4000      304     3696
         Disk usage for Chunk 3                    Total Pages
         ------------------------------------------------------
         OVERHEAD                                            8
         stores_demo:chrisw.catalog                        296
         FREE                                             3696
```

Figure 10-3. oncheck -pe Output That Shows Blobspace Use

Monitoring Simple Large Objects in a Dbspace with oncheck -pT

Use **oncheck -pT** to monitor dbspaces to determine the number of dbspace pages that TEXT and BYTE data use.

This command takes a database name or a table name as a parameter. For each table in the database, or for the specified table, the database server displays a general tblspace report.

Following the general report is a detailed breakdown of page use in the extent, by page type. See the **Type** column for information on TEXT and BYTE data.

The database server can store more than one simple large object on the same blobpage. Therefore, you can count the number of pages that store TEXT or BYTE data in the tblspace, but you cannot estimate the number of simple large objects in the table.

Figure 10-4 on page 10-41 shows sample output.

```
TBLSpace Usage Report for mydemo:chrisw.catalog

    Type                 Pages      Empty  Semi-Full       Full  Very-Full
    ---------------- ---------- ---------- ---------- ---------- ----------
    Free                     7
    Bit-Map                  1
    Index                    2
    Data (Home)              9
    Data (Remainder)         0          0          0          0          0
    Tblspace BLOBs           5          0          0          1          4
                     ----------
    Total Pages             24

    Unused Space Summary

        Unused data bytes in Home pages               3564
        Unused data bytes in Remainder pages             0
        Unused bytes in Tblspace Blob pages           1430

Index Usage Report for index  111_16 on mydemo:chrisw.catalog

                    Average    Average
    Level    Total No. Keys Free Bytes
    ----- -------- -------- ----------
        1        1       74       1058
    ----- -------- -------- ----------
    Total        1       74       1058

Index Usage Report for index  111_18 on mydemo:chrisw.catalog

                    Average    Average
    Level    Total No. Keys Free Bytes
    ----- -------- -------- ----------
        1        1       74        984
    ----- -------- -------- ----------
    Total        1       74        984
```

Figure 10-4. Sample output from oncheck -pT with TEXT or BYTE data

Monitoring Sbspaces

One of the most important areas to monitor in an sbspace is the metadata page use. When you create an sbspace, you specify the size of the metadata area. Also, any time that you add a chunk to the sbspace, you can specify that metadata space be added to the chunk.

If you attempt to insert a new smart large object, but no metadata space is available, you receive an error. The administrator should monitor metadata space availability to prevent this situation from occurring.

Use the following commands to monitor sbspaces.

Command	Description
onstat -g smb s	Displays the storage attributes for all sbspaces in the system: • sbspace name, flags, owner • logging status • average smart-large-object size • first extent size, next extent size, and minimum extent size • maximum I/O access time • lock mode
onstat -g smb c	Displays for each sbspace chunk the following: • chunk number and sbspace name • chunk size and path name • total user data pages and free user data pages • location and number of pages in each user-data and metadata areas See " Using onstat -g smb c" on page 10-46.
oncheck -ce **oncheck -pe**	Displays the following information about sbspace use: • Names of the tables that store smart-large-object data, by chunk • Number of disk pages (not sbpages) used, by table • Number of free user-data pages that remain, by chunk • Number of reserved user-data pages that remain, by chunk • Number of metadata pages used, by chunk The output provides the following totals: • Total number of used pages for all user-data areas and metadata area. The system adds 53 pages for the reserved area to the totals for the user-data area and metadata area. • Number of free pages that remain in the metadata area • Number of free pages that remain in all user-data areas See "Using oncheck -ce and oncheck -pe" on page 10-43 and "Monitoring the Metadata and User-Data Areas" on page 10-45.
onstat -d	Displays the following information about the chunks in each sbspace: • Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data areas • Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data areas See "Using onstat -d," following.
oncheck -cs **oncheck -ps**	Validates and displays information about the metadata areas for sbspaces. See "Using oncheck -cs" on page 10-44 and "Using oncheck -ps" on page 10-45.
oncheck -cS	Displays information about smart-large-object extents and user-data areas for sbspaces.
oncheck -pS	Displays information about smart-large-object extents, user-data areas, and metadata areas for sbspaces. For more information on **oncheck -cS** and **-pS**, see managing sbspaces in the chapter on table performance considerations in your *IBM Informix Dynamic Server Performance Guide*.

Using onstat -d

Use the **onstat -d** option to display the following information about the chunks in each sbspace:

- Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data area

- Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data area

For an example of **onstat -d** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

To find out the total amount of used space, execute the **oncheck -pe** command. For more information, see "Using oncheck -ce and oncheck -pe."

The **onstat -d** option does not register an sbpage as available until the logical log in which a deletion occurred is backed up and the sbpage is freed. Therefore, if you delete 25 smart large objects and immediately execute **onstat -d**, the newly freed space does not appear in the **onstat** output.

Using oncheck -ce and oncheck -pe

Execute **oncheck -ce** to display the size of each sbspace chunk, the total amount of used space, and the amount of free space in the user-data area. The **oncheck -pe** option displays the same information as **oncheck -ce** plus a detailed listing of chunk use. First the dbspaces are listed and then the sbspaces. The **-pe** output provides the following information about sbspace use:

- Names of the tables that store smart-large-object data, by chunk
- Number of disk pages (*not* sbpages) used, by table
- Number of free user-data pages that remain, by chunk
- Number of metadata pages used, by chunk

The output provides the following totals:

- Total number of used pages for the user-data area, metadata area, and reserved area

  The system adds an extra 53 pages for the reserved area to the totals for the user-data area and metadata area.
- Number of free pages that remain in the metadata area
- Number of free pages that remain in the user-data area

Tip: The **oncheck -pe** option provides information about sbspace use in terms of database server pages, not sbpages.

Figure 10-5 on page 10-44 shows sample output. In this example, the sbspace **s9_sbspc** has a total of 214 used pages, 60 free pages in the metadata area, and 726 free pages in the user-data area.

```
Chunk Pathname                                      Size      Used      Free
      2 /ix/ids9.2/./s9_sbspc                         1000       940        60

    Description                                       Offset     Size
    ------------------------------------------------ -------- --------
    RESERVED PAGES                                           0        2
    CHUNK FREELIST PAGE                                      2        1
    s9_sbspc:'informix'.TBLSpace                             3       50
    SBLOBSpace LO [2,2,1]                                   53        8
    SBLOBSpace LO [2,2,2]                                   61        1
...
SBLOBSpace LO [2,2,79]                                     168        1
    SBLOBSpace FREE USER DATA                              169      305
    s9_sbspc:'informix'.sbspace_desc                       474        4
    s9_sbspc:'informix'.chunk_adjunc                       478        4
    s9_sbspc:'informix'.LO_hdr_partn                       482        8
    s9_sbspc:'informix'.LO_ud_free                         490        5
    s9_sbspc:'informix'.LO_hdr_partn                       495       24
    FREE                                                   519       60
    SBLOBSpace FREE USER DATA                              579      421

    Total Used:                                    214
    Total SBLOBSpace FREE META DATA:                60
    Total SBLOBSpace FREE USER DATA:               726
```

Figure 10-5. oncheck -pe Output That Shows Sbspace Use

You can use CHECK EXTENTS as the SQL administration API *command* equivalent to **oncheck -ce**. For information on using SQL API commands, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1 and the *IBM Informix Dynamic Server Administrator's Reference*.

Using oncheck -cs

The **oncheck -cs** and the **oncheck -Cs** options validate the metadata area of an sbspace. Figure 10-6 shows an example of the **-cs** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** checks and displays the metadata for all sbspaces.

Use the **oncheck -cs** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area. To find the number of used pages in the metadata area, total the numbers in the **Used** column. To find the number of free pages in the metadata area, total the numbers in the **Free** column.

For example, based on the field values displayed in Figure 10-6, the total number of used pages in the metadata area for **s9_sbspc** is 33 two-kilobyte pages (or 66 kilobytes). The metadata area contains a total of 62 free pages (or 124 kilobytes).

```
Validating space 's9_sbspc' ...

SBLOBspace Metadata Partition                  Partnum      Used      Free
s9_sbspc:'informix'.TBLSpace                   0x200001        6        44
s9_sbspc:'informix'.sbspace_desc               0x200002        2         2
s9_sbspc:'informix'.chunk_adjunc               0x200003        2         2
s9_sbspc:'informix'.LO_hdr_partn               0x200004       21        11
s9_sbspc:'informix'.LO_ud_free                 0x200005        2         3
```

Figure 10-6. oncheck -cs Output

Using oncheck -ps

The **oncheck -ps** option validates and displays information about the metadata areas in sbspace partitions. Figure 10-7 shows an example of the **-ps** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** validates and displays tblspace information for all storage spaces.

To monitor the amount of free metadata space, run the following command:

```
oncheck -ps spacename
```

The **-ps** output includes information about the locking granularity, **partnum**, number of pages allocated and used, extent size, and number of rows in the metadata area. Use the **oncheck -ps** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area.

If you run **oncheck -ps** for the dbspace that contains the tables where the smart large objects are stored, you can find the number of rows in the table.

```
Validating space 's9_sbspc' ...

TBLSpace Report for
    TBLspace Flags                2801       Page Locking
                                             TBLspace use 4 bit bit-maps
                                             Permanent System TBLspace

    Partition partnum             0x200001
    Number of rows                92
    Number of special columns     0
    Number of keys                0
    Number of extents             1
    Current serial value          1
    First extent size             50
    Next extent size              50
    Number of pages allocated     50
    Number of pages used          6
    Number of data pages          0
    Number of rows                0
    Partition lockid              2097153
    Optical Cluster Partnum       -1
    Current SERIAL8 value         1
    Current REFID value           1
    Created                       Thu May 24 14:14:33 2007
```

Figure 10-7. oncheck -ps Output

Monitoring the Metadata and User-Data Areas

The database server reserves 40 percent of the user-data area as a *reserved area.* The database server uses this reserved space for either the metadata or user data. The metadata area gets used up as smart large objects are added to that sbspace. When the database server runs out of metadata or user-data space, it moves a block of the reserved space to the corresponding area.

When all of the reserve area is used up, the database server cannot move space to the metadata area, even if the user-data area contains free space.

1. As you add smart large objects to the sbspace, use **oncheck -pe** or **onstat -g smb c** to monitor the space in the metadata area, user-data area, and reserved area. For an example, see "Using oncheck -ce and oncheck -pe" on page 10-43.

2. Use the message log to monitor metadata stealing.

   The database server prints messages about the number of pages allocated from the reserved area to the metadata area.
3. Add another chunk to the sbspace before the sbspace runs out of space in the metadata and reserved areas.

   For more information, see "Adding a Chunk to an Sbspace" on page 10-24.
4. The database server writes the FREE_RE and CHKADJUP log records when it moves space from the reserve area to the metadata or user-data area.

For more information, see " Sizing Sbspace Metadata" on page 10-24.

Using onstat -g smb c

Use the **onstat -g smb c** option to monitor the amount of free space in each sbspace chunk, and the size in pages of the user-data, metadata, and reserved areas.

The output of this command shows the number of used pages (**usr pgs**) and free pages (**free pg**) in an sbspace chunk. The metadata area **Md** area includes information that shows the starting page offset and the number of pages.

Loading Data Into a Table

You can load data into an existing table in the following ways.

Method to Load Data	TEXT or BYTE Data	CLOB or BLOB Data	Reference
DB–Access LOAD statement	Yes	Yes	LOAD statement in the *IBM Informix Guide to SQL: Syntax*
dbload utility	Yes	Yes	*IBM Informix Migration Guide*
dbimport utility	Yes	Yes	*IBM Informix Migration Guide*
Informix ESQL/C programs	Yes	Yes	*IBM Informix ESQL/C Programmer's Manual*
onload utility	No	No	*IBM Informix Migration Guide*
onpladm utility	Yes, deluxe mode	Yes, deluxe mode	IBM Informix Server Administrator
High-Performance Loader (HPL)	Yes, deluxe mode	Yes, deluxe mode	*IBM Informix High-Performance Loader User's Guide*

Important: The database server does not contain any mechanisms for compressing TEXT and BYTE data after the data has been loaded into a database.

Compression of Row Data and Storage Optimization

You can compress data in tables and table fragments to reduce the amount of needed disk space. You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Before you compress data, you can estimate the amount of space that you can save.

Compressing data, consolidating data, and returning free space have the following benefits:

- Significant savings in disk storage space

- Reduced disk usage for compressed fragments
- Significant saving of logical log usage, which saves additional space and can prevent bottlenecks for high-throughput OLTP after the compression operation is completed.
- Fewer page reads, because more rows can fit on a page
- Smaller buffer pools, because more data fits in the same size pool
- Reduced I/O activity, because:
  - More compressed rows than uncompressed rows fit on a page
  - Log records for insert, update, and delete operations of compressed rows are smaller
- Ability to compress older fragments of time-fragmented data that are not often accessed, while leaving more recent data that is frequently accessed in uncompressed form
- Ability to free space no longer needed for a table
- Faster backup and restore

I/O-bound tables, for example, those with bad cache hit ratios, are good candidates for compression. In OLTP environments, compressing I/O-bound tables can improve performance.

If your applications run with high buffer cache hit ratios and high performance is more important than space usage, you might not want to compress your data, because compression might slightly decrease performance.

Because compressed data covers fewer pages and has more rows per page than uncompressed data, the query optimizer might choose different plans after compression.

If you use Enterprise Replication (ER), compressing data on one replication server does not affect the data on any other replication server.

If you use High-Availability Data Replication (HDR), data that is compressed in the source table will be compressed in the target table. You cannot perform compression operations on an HDR secondary, RS secondary, or SD secondary server, because the HDR target server must have exactly the same data and physical layout as the source server.

You cannot use the onload and onunload utilities to move compressed data from one database to another. You must uncompress data in compressed tables and fragments before using the onload and onunload utilities.

You perform compression-related operations by running SQL statements that contain SQL administration API commands with compression parameters. Compression-related operations include enable compression, estimate_compression, create_dictionary, compress, repack, repack_offline, shrink, uncompress, and uncompress_offline operations.

Before you can compress a table or fragment, you must run an SQL administration API command that enables compression. If compression is enabled, you must follow Dynamic Server reversion procedures to change back to the version of Dynamic Server that does not have the data compression feature, and you must uncompress or drop any compressed tables and fragments.

Compress, repack, repack_offline, uncompress, and uncompress_offline operations can consume large amounts of logs. Configure your logs to be larger if any workload that you expect to run, including but not limited to these compression operations, will consume log files faster than one every 30 seconds.

Do not drop a dbspace that Change Data Capture (CDC) API is using, if the dbspace ever contained compressed tables, because this could delete compression dictionaries that CDC still needs.

The main alternative to compression for storage needs is to buy more physical storage. The main alternative for reducing bottlenecks in IO-bound workloads is to buy more physical memory to enable the expansion of the buffer pools.

Data that You Can Compress

Table or table-fragment data with frequently repeating long patterns is very compressible. Certain types of data, such as text, might be more compressible than other types of data, such as numeric data, because data types like text might contain longer and more frequently repeating patterns. However, you cannot predict a compression ratio based only on the type of data.

For example:

- Text in different languages or character sets might have different compression ratios, even though the text is stored in CHAR or VARCHAR columns.
- Numeric data that consists mostly of zeros might compress well, while more variable numeric data might not compress well.
- Data with long runs of blank spaces compresses well
- Data that has already been compressed using some other algorithm and data that has been encrypted might not compress well.

Dynamic Server can compress any combination of data types, because it treats all data to be compressed as unstructured sequences of bytes. Thus, the server can compress patterns that span columns, for example, in city, state, and zip code combinations. (Dynamic Server uncompresses a sequence of bytes in the same sequence that existed before the data was compressed.)

There are no restrictions on the types of data that can be compressed.

Compression is applied only to the contents of data rows, including any remainder pieces for rows that span pages, and the images of those rows that are contained in logical log records.

Many types of large-object data (such as images and sound samples) in rows might already be compressed, so compressing the data again would not achieve any additional saving of space.

Data that You Cannot Compress

You cannot compress data in indexes, and you cannot compress data in some types of tables and fragments.

You cannot compress data in rows in:

- Tables or fragments in the **sysmaster**, **sysutils**, **sysuser**, **syscdr**, and **syscdcv1** databases
- Catalogs

- Temporary tables
- Virtual-table interface tables
- A tblspace **tblspace** (These are a hidden fragments, one per dbspace. Each one holds metadata about all of the fragments in the dbspace.)
- Internal partition tables
- Dictionary tables, one per dbspace (These tables hold compression dictionaries for the fragments or tables that are compressed in that dbspace and metadata about the dictionaries.)
- Indexes

Compression is not applied to index data, LOB data that is stored outside of the row, or any other form of non-row data.

Encrypted data, data that is already compressed by another algorithm, and data without long repeating patterns compresses poorly or does not compress. Try to avoid placing columns with data that compresses poorly between columns that have frequent patterns to prevent the potential disruption of column-spanning patterns that can be compressed.

If XML data is stored with the first portion in a row and the remainder outside of the row, compression is only applied to the portion that is stored in the row.

Dynamic Server compresses images of the rows only if the images of the compress rows will be smaller than the uncompressed images. Even if compressed rows are only slightly smaller than their uncompressed images, a small saving of space can enable the server to put more rows onto pages.

Compression Ratios

The compression ratio depends on the data being compressed. The compression algorithm that Dynamic Server uses is a dictionary-based algorithm that performs operations on the patterns of the data that were found to be the most frequent, weighted by length, in the data that was sampled at the time the dictionary was built.

If the typical data distribution skews away from the data that was sampled when the dictionary was created, compression ratios can decrease.

The maximum possible compression ratio is 90 percent. This is because the maximum possible compression of any sequence of bytes occurs by replacing each group of 15 bytes with a single 12-bit symbol number, yielding a compressed image that is ten percent of the size of the original image. However, the 90 percent ratio is never quite achieved, because Dynamic Server adds a single byte of metadata to each compressed image.

Compression Estimates

Before you compress a table or table fragment, you can estimate the amount of space you can save if data is compressed. The ratios you display are estimates based on samples of row data. The actual ratio of saved space could vary slightly.

Dynamic Server estimates the compression ratios by random sampling of row data (using the same sampling algorithm as dictionary building) and then summing up the sizes of the following items:

- Uncompressed row images

- Compressed row images using a new compression dictionary (which is temporarily created by the estimate compression command)
- Compressed row images using the existing dictionary if there is one (If there is no existing dictionary, this value is the same as the sum of sizes of uncompressed row images.)

The actual space saving ratios achieved might vary for these reasons:

- A small sampling error can occur.
- The estimates are based on raw compressibility of the rows.

  For example, the server generally tries to put the entire row onto a single page. So, if each uncompressed row nearly fills a complete page and the compression ratio is less than 50 percent, the compressed rows will still fill more than half a page each and the server will tend to put each row on a separate page even after compression. In this case, although the estimated compression ratio could be something like 45 percent, the actual space savings might turn out to be 0 percent.

  Uncompressed rows fill slightly more than half a page each. Thus, each uncompressed row will actually consume a full page since two full rows do not fit. For example, the estimated compression ratio could be something like 5 percent, but this could be just enough to shrink the rows to be less than half a page each. Thus, after compression, two rows would fit on a page and the true space savings could be 50 percent.

The actual compression achieved might also vary from the estimate because Dynamic Server can never store more than 255 rows on a single page. Thus, small rows or large pages can reduce the total savings that compression can achieve. For example, if 200 rows fit onto a page before compression, no matter how small the rows are when compressed, the maximum effective compression ratio is approximately 20 percent, because only 255 rows can fit on a page after compression.

If you are using a page size that is larger than the minimum page size, one way to increase the realized compression space savings is to switch to smaller pages, so that:

- The 255 row limit can no longer be reached.
- If this limit is still reached, there is less unused space on the pages.

More (or less) space can be saved, compared to the estimate, if the compress operation is combined with a repack operation, shrink operation, or repack and shrink operation. The repack operation can save additional space only if more compressed rows fit on a page than uncompressed rows. The shrink operation can save space at the dbspace level if the repack operation frees space.

Compression Dictionaries

A separate compression dictionary exists for each compressed fragment and each compressed non-fragmented table. Each compression dictionary is a library of frequently occurring patterns in the fragment or table data and the symbol numbers that replace the patterns.

A compression dictionary is built using data that is sampled randomly from a fragment or non-fragmented table that contains at least 2,000 rows. If the fragment or table does not contain 2,000 rows, Dynamic Server will not build a compression dictionary.

The compression dictionary can store a maximum of 3,840 patterns, each of which can be from two to 15 bytes in length. (Patterns that are longer than seven bytes reduce the total number of patterns that the dictionary can hold.) Each of these patterns is represented by a 12-bit symbol number in a compressed row. To be compressed, a sequence of bytes in the input row image must exactly match a complete pattern in the dictionary. A row that does not have enough pattern matches against the dictionary might not be compressible, because each byte of an input row that did not completely match is replaced in the compressed image by 12 bits (1.5 bytes).

Dynamic Server attempts to capture the best compressible patterns (the frequency of the pattern multiplied by the length). Data is compressed by replacing occurrences of the patterns with the corresponding symbol numbers from the dictionary, and replacing occurrences of bytes that do not match any pattern with special reserved symbol numbers.

All dictionaries for the tables or fragments in a dbspace are stored in a hidden dictionary table in that dbspace. The **syscompdicts_full** table and the **syscompdicts** view in the **sysmaster** database provide information on the compression dictionaries.

Typically, approximately 100 KB of space is required for storing the compression dictionary for a compressed fragment or table. Thus, very small tables are not good candidates for compression, because you might not be able to gain back enough space from compressing the rows to offset the storage cost of the compression dictionary.

Additionally, Dynamic Server cannot compress an individual row to be smaller than four bytes long. This is because the server must leave room in case the row image later grows beyond what the page can hold. Therefore, you should not try to compress fragments or non-fragmented tables with rows that contain four bytes or are shorter than four bytes.

Compression Information that You Can View

You can use Dynamic Server utilities, a **sysmaster** database table, and a **sysmaster** view to display compression statistics, information about compression dictionaries, and the compression dictionary.

Table 10-1. Utilities and the **sysmaster** *Table and View that Show Compression Information*

Utilities, Table, or View	Description
oncheck -pT option	Displays the number of any compressed rows in a table or table fragment and the percentage of table or table-fragment rows that are compressed. If table or fragment rows are not compressed, the ″Compressed Data Summary″ section does not appear in the output.
onlog -c option	Uses the compression dictionary to expand compressed data and display the uncompressed contents of compressed log records.
onstat –g dsk option	Displays information that shows the progress of currently running compression operations.

Table 10-1. Utilities and the **sysmaster** *Table and View that Show Compression Information (continued)*

Utilities, Table, or View	Description
onstat -g ppd option	Displays information on the active compression dictionaries that exist for currently open compressed fragments (also referred to as partitions). This option shows the same information as the **syscompdicts** view in the **sysmaster** database.
syscompdicts_full table in the **sysmaster** database	Displays metadata about the compression dictionary and the compression dictionary binary object. Only user **informix** can access this table.
syscompdicts view in the **sysmaster** database	Displays the same information as the **syscompdicts_full** table, except that for security reasons, it excludes the **dict_dictionary** column, which contains the compression dictionary binary object.

You can use an UNLOAD statement to unload the compression dictionary from the **syscompdicts_full** table to the compression dictionary file, as follows:

```
UNLOAD TO 'compression_dictionary_file'
   SELECT * FROM sysmaster:syscompdicts_full;
```

For more information about the **onlog** Utility, the **onstat -g dsk** option, the **onstat -g ppd** option, the **oncheck -pT** option, the **syscompdicts_full** table, and the **syscompdicts** view, see the *IBM Informix Dynamic Server Administrator's Reference*.

Illustration of Compressed Data and Storage Optimization

The illustration in this topic shows uncompressed data that uses most of the space in a fragment, free space that is created when the data is compressed, free space that is moved to the end of the fragment after a repack operation, and data that remains in the fragment after a shrink operation.

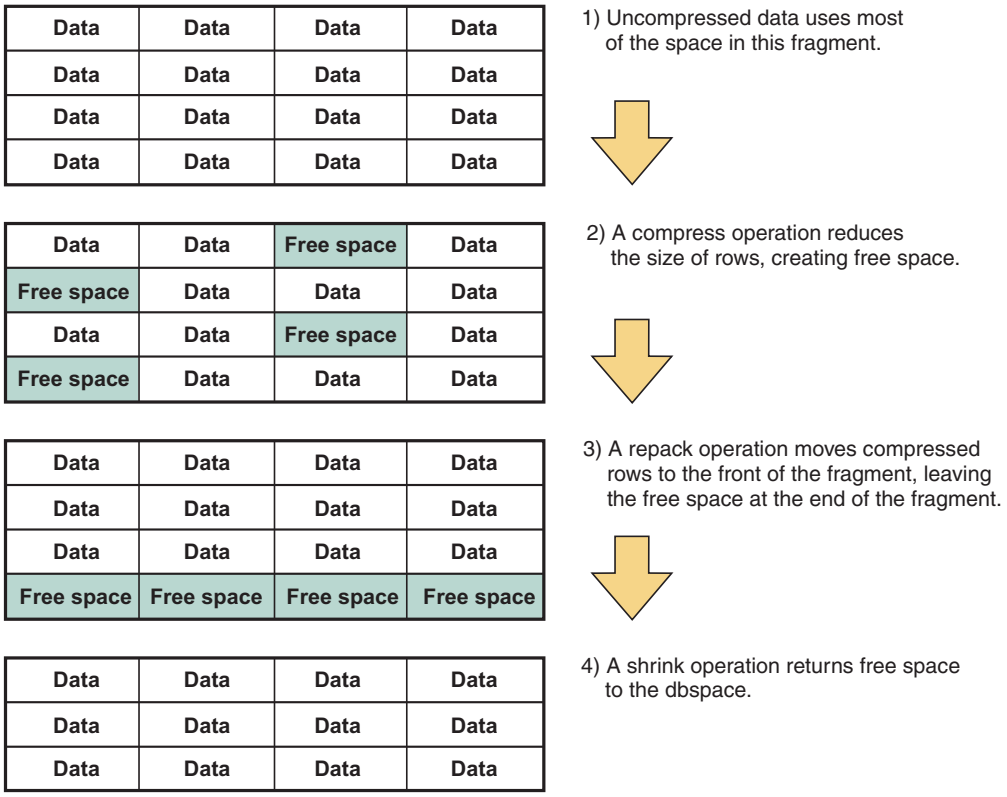


Figure 10-8. Data in a Fragment during the Compression and Storage Optimization Process

Compressing and Uncompressing Row Data

This scenario shows how you can run SQL administration API commands to administer compression and storage optimization. In this scenario, there is a table named **rock** in a database named **music** owned by user **mario**.

Prerequisites:

- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Compress, repack, repack_offline, uncompress, and uncompress_offline operations can consume large amounts of logs. Configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to these compression operations, will consume log files faster than one every 30 seconds.

To compress and uncompress row data:

1. You expect to compress some of the data in your database, so you run the followings command once to enable compression for the database server:

   ```
   EXECUTE FUNCTION task("enable compression");
   ```

   You must enable compression before you can compress data. However, you do not need to enable compression to use the **estimate_compression** argument to estimate how much space compressing your data would save. You also do not need to enable compression if you want to use the **shrink**, **repack**, or **repack_offline** arguments to free up space from tables without compressing any row data.

2. You are not sure if you want to compress the **rock** table, so you run the following command to check how much space you could save by compressing the table:

   ```
   EXECUTE FUNCTION task("table estimate_compression", "rock", "music", "mario");
   ```

   You review the resulting report, which indicates you could save 75 percent of the space currently used by the **rock** table. You decide to compress the table.
3. Before you compress data, you want to create a compression dictionary, which contains information that Dynamic Server uses to compress data in the **rock** table. You run the following command

   ```
   EXECUTE FUNCTION task("table create_dictionary", "rock", "music", "mario");
   ```

   If you do not create the compression dictionary as a separate step, Dynamic Server creates the dictionary automatically when you compress data.
4. You decide that you want to compress data in the **rock** table, consolidate the data, and then return the free space to the dbspace. You run the following command:

   ```
   EXECUTE FUNCTION task("table compress repack shrink", "rock", "music", "mario");
   ```

   After the existing rows are compressed, Dynamic Server consolidates the free space that is left at the end of the table, and then removes the free space from the table, returning that space to the dbspace.
5. Now suppose that you need to uncompress the data. You run the following command:

   ```
   EXECUTE FUNCTION task("table uncompress", "rock", "music", "mario");
   ```
6. You want to remove the compression dictionary.
   a. Verify Enterprise Replication (ER) does not need the dictionary.

      Do not remove compression dictionaries for uncompressed or dropped tables and fragments, if you do need the dictionaries for ER.
   b. Archive the dbspace that contains the table or fragment with a compression dictionary.
   c. Run this command:

      ```
      EXECUTE FUNCTION task("table purge_dictionary", "rock", "music", "mario");
      ```

You compress and uncompress data in table fragments the same way you compress and uncompress data in rows, except that the commands you run have the following format:

```
EXECUTE FUNCTION task("fragment compression_arguments", "partnum_list");
```

For more information on the syntax for the SQL administration API compression commands, see the *IBM Informix Dynamic Server Administrator's Reference*.

Enabling Compression

Before you can compress or uncompress tables or table fragments, you must run the SQL administration API admin() or task() command that enables compression. Running the command for enabling compression is required before you can perform the first compress or uncompress operation in an instance.

However, you can estimate compression ratios, consolidate free space (repack), and return free space to a table or fragment (shrink) before you enable compression.

Prerequisite: You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.

To enable compression:

Run the admin() or task() function with the **enable compression** argument. For example, specify:

```
EXECUTE FUNCTION task("enable compression");
```

After compression is enabled, you can create a compression dictionary, or you can compress the rows of a table or the rows of a particular fragment or all fragments of a fragmented table.

Estimating Compression Ratios

You can estimate the percentage of space that you can save if you compress tables or specific or all fragments of fragmented tables. The command displays information that you can use to determine if you want to compress or recompress row data.

The command that estimates compression ratios always estimates both a new compression ratio and a current ratio.

For general information on compression ratios and estimates, see "Compression Ratios" on page 10-49 and "Compression Estimates" on page 10-49.

Prerequisite: You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.

Compression does not need to be enabled to estimate compression benefits.

To estimate the compression benefit:

Run the admin() or task() function with the **estimate_compression** argument. For example, for a table use this syntax:

```
EXECUTE FUNCTION task("table estimate_compression",
"table_name", "database_name", "owner_name");
```

For a fragment, use this syntax:

```
EXECUTE FUNCTION task("fragment estimate_compression",
"partnum_list");
```

The following example shows a command that tells Dynamic Server to estimate the benefit of compressing a table named "cash_transaction" in a "store123" database in which "wong" is the owner.

```
EXECUTE FUNCTION task("table estimate_compression", "cash_transaction",
 "store123", "wong");
```

The estimate_compression operation displays the estimated compression ratio that can be achieved, the current compression ratio, an estimate of the percentage gain or loss, the partition number of each fragment, and the full name of the table, including the database, owner, and table names. The current ratio is 0.0 percent if the table is not compressed.

In the following example, the first fragment is already compressed. The second fragment is not compressed. If you recompress the first fragment, a .4 percent increase in saved space could occur. If you compress the second fragment, a 75.7 percent increase could occur.

```
  est   curr  change    partnum     table
  ----- ----- ------ -------------  ---------------------------------
  75.7% 75.3%   +0.4    0x00200003  store123:wong.cash_transaction
  75.7%  0.0%  +75.7    0x00300002  store123:wong.cash_transaction
```

Output from compression estimates for tables and fragments looks the same, except that the output for a table always shows all fragments in the table, while the output for a fragment only shows information for the specified fragments.

Creating a Compression Dictionary

You can create a compression dictionary based on existing rows for Dynamic Server to use when compressing data in tables or table fragments. After you create the dictionary, Dynamic Server will use the dictionary to compress newly inserted or updated rows.

If a compression dictionary does not exist, you can also create one when you run the compress command. The only difference between the two commands is that the compress command also compresses existing data in the table or fragment.

For general information on compression dictionaries, see "Compression Dictionaries" on page 10-50.

Prerequisites:

- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- If you want to create a compression dictionary for a fragment, the fragment must contain at least 2,000 rows. If you want to create a compression dictionary for a table, each fragment of the table must contain at least 2,000 rows.

To create a compression dictionary:

Run the admin() or task() function with the **table create_dictionary** or **fragment create_dictionary** arguments.
For example:

- For a table, specify information as follows:

  ```
  EXECUTE FUNCTION task("table create_dictionary", "table_name",
  "database_name", "owner_name");
  ```

  The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Dynamic Server uses the current database and owner name.
- For a fragment, specify information as follows:

  ```
  EXECUTE FUNCTION task("fragment create_dictionary", "partnum_list");
  ```

  The *partnum_list* is a space-separated list of partition numbers.

The following example shows a command that tells Dynamic Server to create a compression dictionary for a table named ″classical″ in a ″music″ database in which ″shakar″ is the owner.

```
EXECUTE FUNCTION task("table create_dictionary","classical","music","shakar");
```

To compress data in existing table or fragment rows after you create the compression dictionary, you must run a compress command.

You can delete a compression dictionary only after you uncompress the table or fragment.

Compressing Data in Tables and Table Fragments

You can compress data in tables and fragments with an SQL administration API admin() or task() commands. The compress operation creates a compression dictionary if one does not exist, and it compresses rows without moving them.

Prerequisites:

- There must be at least 2,000 rows in each fragment of the table, not just a total of 2,000 rows in the table as a whole.
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- You must configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to a compress operation, will consume log files faster than one every 30 seconds.

To compress a table or fragment:

1. Run the admin() or task() function with the **table compress** or **fragment compress** command arguments.

   For example, for a table specify:

   ```
   EXECUTE FUNCTION task("table compress", "table_name",
   "database_name", "owner_name");
   ```

   The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Dynamic Server uses the current database and owner name.

   For example, for a fragment specify:

   ```
   EXECUTE FUNCTION task("fragment compress", "partnum_list");
   ```

   The *partnum_list* is a space-separated list of partition numbers that belong to the same table.
2. Optionally expand the arguments to include **repack** and **shrink** in any of the following combinations:
   - **compress repack**
   - **compress repack shrink**
   - **compress shrink**

The following example shows a command that tells Dynamic Server to compress a table named ″opera″ in a ″music″ database in which ″bob″ is the owner.

```
EXECUTE FUNCTION task("table compress","opera","music","bob");
```

The following example shows a command that tells Dynamic Server to compress the fragment with the partition number 14680071.

```
EXECUTE FUNCTION admin("fragment compress","14680071");
```

If you interrupt a compress operation and reissue a compress command, Dynamic Server continues compressing, using the existing compression dictionary.

If you want to consolidate free space or return free space without compressing or recompressing, you can run a command that tells the server to repack, shrink, or repack and shrink.

If you change the fragmentation strategy for a table after you perform a compression operation, you will need to recompress.

You can cancel a command with a **compress** argument, for example, by typing CTRL-C in DB-Access. You can reissue commands with **repack** or **repack_offline** arguments after a prior interrupted command. (Compress and repack operations are logged, but run in small portions.)

For more information on command syntax and details on the compression arguments, see *IBM Informix Dynamic Server Administrator's Reference*.

Consolidating Free Space in Tables

You can consolidate (repack) free space in tables and fragments when you compress the tables or fragments or separately without compressing.

You can perform a repack operation online or offline, using the **repack** or **repack_offline** argument. The repack_offline operation is the same as the repack operation, except that Dynamic Server performs the operation while holding an exclusive lock on the table or fragment. This operation prevents all other access to data until the operation is completed.

If light appends occur in a table or fragment while a repack operation is occurring, the repack operation will not complete the consolidation of space at the end of a table or fragment. The repack operation does not complete because the new extents are added in the location where the repack operation has already occurred, so space cannot be returned to the dbspace. To complete the repack process, you must run a second repack operation after light append activity has completed. This second repack operation builds on the work of the first repack operation.

Dropping or disabling indexes before you compete a repack_offline operation can decrease the amount of time that it takes the server to complete the operation. Afterwards, you can re-create or re-enable the indexes, preferably taking advantage of PDQ. Dropping or disabling the indexes and then creating or enabling them again can be faster than completing a repack_offline operation without doing this.

Prerequisites:

- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to a repack or repack_offline operation, will consume log files faster than one every 30 seconds.

To consolidate free space in a table:

1. Run the admin() or task() function with the **table repack**, **table repack_offline**, **fragment repack** or **fragment repack_offline** command arguments.

   For example, for a table specify:

   ```
   EXECUTE FUNCTION task("table repack", "table_name",
   "database_name", "owner_name");
   ```

   The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Dynamic Server uses the current database and owner name.

   For example, for a fragment specify:

   ```
   EXECUTE FUNCTION task("fragment repack_offline", "partnum_list");
   ```

   The *partnum_list* is a space-separated list of partition numbers that belong to the same table.

2. Optionally expand the arguments to include **compress** and **shrink** in any of the following combinations:
   - **compress repack**
   - **compress repack shrink**
   - **repack shrink**

The following example shows a command that tells Dynamic Server to consolidate free space in a table named ″opera″ in a ″music″ database in which ″bob″ is the owner.

```
EXECUTE FUNCTION task("table repack","opera","music","bob");
```

The following example shows a command that tells Dynamic Server to consolidate free space offline in a table named ″folk″ in a ″music″ database in which ″janna″ is the owner.

```
EXECUTE FUNCTION task("table repack_offline","folk","music","janna");
```

The following example shows a command that tells the Dynamic Server to consolidate free space and return the space to the dbspace in a fragment with a partition number of 14680071.

```
EXECUTE FUNCTION task("fragment repack shrink", "14680071");
```

You can cancel a command with a **compress** argument, for example, by typing CTRL-C in DB-Access. You can reissue commands with **repack** or **repack_offline** arguments after a prior interrupted command. (Compress and repack operations are logged, but run in small portions.)

Returning Free Space to the dbspace

You can return free space to the dbspace (shrink the space) when you compress, repack, or compress and repack tables or fragments; or you can return free space separately without compressing or repacking. Returning free space reduces the total size of the fragment or table.

You can safely shrink the entire table without compromising the allocation strategy of the table. For example, if you have a fragmented table with one fragment for each day of the week and many fragments pre-allocated for future use, you can shrink the table without compromising this allocation strategy. If the table is empty, Dynamic Server shrinks the table to the initial extent size specified when the table was created.

When you initiate a shrink operation, Dynamic Server shortens extents as follows:

- It shortens all extents except the first extent to as small a size as possible.
- If the table is entirely in the first extent (for example, because the table is an empty table), Dynamic Server does not shrink the first extent to a size that was smaller than the extent size that was specified when the table was created with the CREATE TABLE statement.

You can use the MODIFY EXTENT SIZE clause of the ALTER TABLE statement to reduce the current extent size. After you do this, you can rerun the shrink operation to shrink the first extent to the new extent size.

Prerequisite: You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.

To return free space to the dbspace:

1. Run the admin() or task() function with the **table shrink** or **fragment shrink** arguments.

   For example, for a table specify:

   ```
   EXECUTE FUNCTION admin("table shrink", "table_name",
   "database_name", "owner_name");
   ```

   The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Dynamic Server uses the current database and owner name.

   For example, for a fragment specify:

   ```
   EXECUTE FUNCTION task("fragment shrink", "partnum_list");
   ```

   The *partnum_list* is a space-separated list of partition numbers that belong to the same table.
2. Optionally expand the arguments to include **compress** and **repack** in any of the following combinations:
   - **compress repack shrink**
   - **compress shrink**
   - **repack shrink**

The following example shows a command that tells Dynamic Server to shrink a table named ″opera″ in a ″music″ database in which ″bob″ is the owner.

```
EXECUTE FUNCTION task("table shrink","opera","music","bob");
```

The following example shows a command that tells the Dynamic Server to repack and shrink a fragment with a partition number of 14680071.

```
EXECUTE FUNCTION task("fragment repack shrink," "14680071");
```

Uncompressing Data

You can uncompress previously compressed tables and fragments. Uncompressing a table or fragment deactivates compression for new insert and update operations, uncompresses all compressed rows, deactivates the compression dictionary, and allocates new pages for rows that no longer fit on their original pages.

You can uncompress online or offline, using the **uncompress** or **uncompress_offline** argument. An uncompress_offline operation is the same as the uncompress operation, except that this operation is performed while holding an exclusive lock on the fragment, preventing all other access to the fragment data until the operation is completed

Dropping or disabling indexes before you compete an uncompress_offline operation can decrease the amount of time that it takes the server to complete the operation. Afterwards, you can re-create or re-enable the indexes, preferably taking advantage of PDQ. Dropping or disabling the indexes and then creating or enabling them again can be faster than completing a repack_offline or uncompress_offline operation without doing this.

Prerequisites:

- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- The table or fragments must be compressed.

- Configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to an uncompress or uncompress_offline operation, will consume log files faster than one every 30 seconds.

To uncompress data in a table or fragment:

Run the admin() or task() function with the **table uncompress**, **table uncompress_offline**, **fragment uncompress**, or **fragment uncompress_offline** command arguments.
For a table, specify information as follows:

```
EXECUTE FUNCTION task("table uncompress", "table_name",
"database_name", "owner_name");
```

or

```
EXECUTE FUNCTION admin("table uncompress_offline",
"table_name", "database_name", "owner_name");
```

The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Dynamic Server uses the current database and owner name.
For a fragment, specify information as follows:

```
EXECUTE FUNCTION task("fragment uncompress", "partnum_list");
```

or

```
EXECUTE FUNCTION task("fragment uncompress_offline", "partnum_list");
```

Examples

The following example shows a command that tells Dynamic Server to uncompress a table named ″rock″ in a ″music″ database in which ″mario″ is the owner.

```
EXECUTE FUNCTION task("table uncompress","rock","music","mario");
```

The following example shows a command that tells the Dynamic Server to uncompress offline a fragment with a partition number of 14680071.

```
EXECUTE FUNCTION task("fragment uncompress_offline," "14680071");
```

If a table is uncompressed, Dynamic Server marks the dictionary for that table as inactive. Dynamic Server does not delete dictionaries, because Enterprise Replication functions use the dictionaries for older logs. You can delete the dictionaries that you no longer need.

You can cancel a command with an **uncompress** argument, for example, by typing CTRL-C in DB-Access.

You can reissue commands with **uncompress**and **uncompress_offline** arguments after a prior interrupted command. (Compress, repack, and uncompress operations are logged, but run in small portions.)

Deleting Compression Dictionaries

You can delete an inactive compression dictionary for a specific table or fragment, you can delete all inactive compression dictionaries, or you can delete all inactive compression dictionaries up to a specified date. You must uncompress tables and fragments, which makes the dictionaries inactive, before you delete any compression dictionaries that were created for the tables and fragments.

Do not remove compression dictionaries that Enterprise Replication needs.

Prerequisites:

- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Uncompress or drop the table or fragment before deleting the associated dictionary. You only can delete a compression dictionary that is no longer used by a compressed table or fragment.
- Be sure that Enterprise Replication functions are not using the compression dictionary for older logs.
- Archive any dbspace that contains a table or fragment with a compression dictionary, even if you uncompressed data in the table or fragment and the dictionary is no longer active.

To delete one or more compression dictionaries that you no longer need:

Run the admin() or task() function with:

- The **table purge_dictionary** or **fragment purge_dictionary** command arguments to delete a specific, inactive dictionary.
- The **compression purge_dictionary** command arguments to delete all dictionaries.
- The **compression purge_dictionary** command arguments and a date to delete all dictionaries that were created before and on the date.

  You can use any date in a format that can be converted to a DATE data type based on your locale and environment. For example, you can specify `01/31/2009`, `01/31/09`, or `Jan 31, 2009`.

Examples

Run the following command to remove the inactive dictionary for a table named ″latin″ in a ″music″ database in which ″arlette″ is the owner.

```
EXECUTE FUNCTION task("table purge_dictionary",
"latin", "music", "arlette");
```

Run the following command to remove all dictionaries that were created before and on March 8, 2009:

```
EXECUTE FUNCTION task("compression purge_dictionary", "03/08/09");
```

Moving Compressed Data

You can use the High-Performance Loader (HPL) or any of the other Dynamic Server utilities, except the **onunload** and **onload** utilities, to move compressed data.

You cannot use the **onload** and **onunload** utilities to move compressed data from one database to another. You must uncompress data in compressed tables and fragments before using the **onload** and **onunload** utilities.

If you use the **dbimport** and **dbexport** utilities to move compressed data, you must manually re-enable compression after you import the data. Do this because importing does not preserve enable compression option.

For details on using HPL or the Dynamic Server utilities, see the *IBM Informix High-Performance Loader User's Guide* or the *IBM Informix Migration Guide*.

Part 3. Logging and Log Administration

Chapter 11. Logging

In This Chapter

This chapter describes logging of Informix databases and addresses the following questions:

- Which database server processes require logging?
- What is transaction logging?
- What database server activity is logged?
- What is the database-logging status?
- Who can set or change the database logging status?

All the databases managed by a single database server instance store their log records in the same logical log, regardless of whether they use transaction logging. Most database users might be concerned with whether transaction logging is buffered or whether a table uses logging.

If you want to change the database-logging status, see "Settings or Changes for Logging Status or Mode" on page 11-7.

Database Server Processes That Require Logging

As Dynamic Server operates, processing transactions, tracking data storage, and ensuring data consistency, Dynamic Server automatically generates *logical-log records* for some of the actions that it takes. Most of the time the database server makes no further use of the logical-log records. However, when the database server needs to roll back a transaction, to execute a fast recovery after a system failure, for example, the logical-log records are critical. The logical-log records are at the heart of the data-recovery mechanisms.

The database server stores the logical-log records in a *logical log*. The logical log is made up of *logical-log files* that the database server manages on disk until they have been safely transferred offline (*backed up*). The database server administrator keeps the backed up logical-log files until they are needed during a data restore, or until the administrator decides that the records are no longer needed for a restore. See Chapter 13, "Logical Log," on page 13-1 for more information on logical logs.

The logical-log records themselves are variable length. This arrangement increases the number of logical-log records that can be written to a page in the logical-log buffer. However, the database server often flushes the logical-log buffer before the page is full. For more information on the format of logical-log records, see the chapter on interpreting logical-log records in the *IBM Informix Administrator's Reference*.

The database server uses logical-log records when it performs various functions that recover data and ensure data consistency, as follows:

- **Transaction rollback**. If a database is using transaction logging and a transaction must be rolled back, the database server uses the logical-log records to reverse the changes made during the transaction. For more information, see "Transaction Logging" on page 11-2.

- **Fast recovery**. If the database server shuts down in an uncontrolled manner, the database server uses the logical-log records to recover all transactions that occurred since the oldest update not yet flushed to disk and to roll back any uncommitted transactions. (When all the data in shared memory and on disk are the same, they are *physically consistent*.) The database server uses the logical-log records in fast recovery when it returns the entire database server to a state of logical consistency up to the point of the most recent logical-log record. (For more information, see "Fast Recovery After a Checkpoint" on page 15-8.)
- **Data restoration**. The database server uses the most recent storage-space and logical-log backups to re-create the database server system up to the point of the most recently backed-up logical-log record. The logical restore applies all the log records since the last storage-space backup.
- **Deferred checking**. If a transaction uses the SET CONSTRAINTS statement to set checking to DEFERRED, the database server does not check the constraints until the transaction is committed. If a constraint error occurs while the transaction is being committed, the database server uses logical-log records to roll back the transaction. For more information, see SET Database Object Mode in the *IBM Informix Guide to SQL: Syntax*.
- **Cascading deletes**. Cascading deletes on referential constraints use logical-log records to ensure that a transaction can be rolled back if a parent row is deleted and the system fails before the children rows are deleted. For information on table inheritance, see the *IBM Informix Database Design and Implementation Guide*. For information on primary key and foreign key constraints, see the *IBM Informix Guide to SQL: Tutorial*.
- **Distributed transactions**. Each database server involved in a distributed transaction keeps logical-log records of the transaction. This process ensures data integrity and consistency, even if a failure occurs on one of the database servers that is performing the transaction. For more information, see "Two-Phase Commit and Logical-Log Records" on page 25-15.
- **Data Replication**. Data Replication environments that use HDR secondary, SD secondary, and RS secondary servers use logical-log records to maintain consistent data on the primary and secondary database servers so that one of the database servers can be used quickly as a backup database server if the other fails. For more details, see "How Data Replication Works" on page 19-6.
- **Enterprise Replication**. You must use database logging with Enterprise Replication because it replicates the data from the logical-log records. For more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Transaction Logging

A database or table is said to *have* or *use* transaction logging when SQL data manipulation statements in a database generate logical-log records.

The database-logging *status* indicates whether a database uses transaction logging. The *log-buffering mode* indicates whether a database uses buffered or unbuffered logging, or ANSI-compliant logging. For more information, see "Database-Logging Status" on page 11-5 and Chapter 12, "Managing the Database-Logging Mode," on page 12-1.

When you create a database, you specify whether it uses *transaction logging* and, if it does, what log-buffering mechanism it uses. After the database is created, you can turn off database logging or change to buffered logging, for example. Even if you turn off transaction logging for all databases, the database server always logs

some events. For more information, see "Activity That is Always Logged" and "Database Logging in an X/Open DTP Environment" on page 11-7.

You can use logging or nonlogging tables within a database. The user who creates the table specifies the type of table. Even if you use nonlogging tables, the database server always logs some events. For more information, see "Table Types for Dynamic Server" on page 9-23.

Logging of SQL Statements and Database Server Activity

Three types of logged activity are possible in the database server:

- Activity that is always logged
- Activity that is logged only for databases that use transaction logging
- Activity that is never logged

Activity That is Always Logged

Some database operations always generate logical-log records, even if you turn off transaction logging or use nonlogging tables.

The following operations are always logged for permanent tables:

- Certain SQL statements, including SQL data definition statements
- Storage-space backups
- Checkpoints
- Administrative changes to the database server configuration such as adding a chunk or dbspace
- Allocation of new extents to tables
- A change to the logging status of a database
- Smart-large-object operations:
  - Creating
  - Deleting
  - Allocating and deallocating extents
  - Truncating
  - Combining and splitting chunk free list pages
  - Changing the LO header and the LO reference count
- Sbspace metadata
- Blobspaces

The table below lists statements that generate operations that are logged.

ALTER ACCESS_METHOD	CREATE ROUTINE	DROP SEQUENCE
ALTER FRAGMENT	CREATE ROUTINE FROM	DROP SYNONYM
ALTER FUNCTION	CREATE ROW TYPE	DROP TABLE
ALTER INDEX	CREATE SCHEMA	DROP TRIGGER
ALTER PROCEDURE	CREATE SEQUENCE	DROP TYPE
ALTER ROUTINE	CREATE SYNONYM	DROP VIEW
ALTER SEQUENCE	CREATE TABLE	CREATE XADATASOURCE
ALTER TABLE	CREATE Temporary TABLE	DROP XADATASOURCE TYPE
CLOSE DATABASE	CREATE TRIGGER	GRANT
CREATE ACCESS_METHOD	CREATE VIEW	RENAME COLUMN
CREATE AGGREGATE	CREATE XADATASOURCE	RENAME DATABASE
CREATE CAST	DROP ACCESS_METHOD	RENAME INDEX
CREATE DATABASE	DROP AGGREGATE	RENAME SEQUENCE
CREATE DISTINCT TYPE	DROP CAST	RENAME TABLE
CREATE EXTERNAL TABLE	DROP DATABASE	REVOKE
CREATE FUNCTION	DROP FUNCTION	REVOKE FRAGMENT
CREATE FUNCTION FROM	DROP INDEX	
CREATE INDEX	DROP OPCLASS	
CREATE OPAQUE TYPE	DROP PROCEDURE	
CREATE OPCLASS	DROP ROLE	
CREATE PROCEDURE	DROP ROUTINE	
CREATE PROCEDURE FROM	DROP ROW TYPE	
CREATE ROLE	DROP XADATASOURCE TYPE	

Activity Logged for Databases with Transaction Logging

If a database uses transaction logging, the following SQL statements generate one or more log records. If these statements are rolled back, the rollback also generates log records.

DELETE	LOAD	UNLOAD
FLUSH	PUT	UPDATE
INSERT	SELECT INTO TEMP	

The following SQL statements generate log records in special situations.

SQL Statement	Log Record That the Statement Generates
BEGIN WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
COMMIT WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
EXECUTE	Whether this statement generates a log record depends on the command being executed.
EXECUTE FUNCTION	Whether this statement generates a log record depends on the function being executed.
EXECUTE IMMEDIATE	Whether this statement generates a log record depends on the command being executed.
EXECUTE PROCEDURE	Whether this statement generates a log record depends on the procedure being executed.

Activity That is Not Logged

The following SQL statements do not produce log records, regardless of the database logging mode.

ALLOCATE COLLECTION	DEALLOCATE ROW	LOCK TABLE
ALLOCATE DESCRIPTOR	DECLARE	OPEN
ALLOCATE ROW	DESCRIBE	OUTPUT
CLOSE	DISCONNECT	PREPARE
CONNECT	FETCH	SELECT
DATABASE	FREE	SET
DEALLOCATE COLLECTION	GET DESCRIPTOR	UNLOCK TABLE
DEALLOCATE DESCRIPTOR	GET DIAGNOSTICS	WHENEVER
DEALLOCATE ROW	INFO	

For temporary tables in temporary dbspaces, nothing is logged, not even the SQL statements listed in "Activity That is Always Logged" on page 11-3. If you include temporary (nonlogging) dbspaces in DBSPACETEMP, the database server places nonlogging tables in these temporary dbspaces first. For more information, see "Temporary Tables" on page 9-32.

Database-Logging Status

You must use transaction logging with a database to take advantage of any of the features listed in "Database Server Processes That Require Logging" on page 11-1.

Every database that the database server manages has a logging status. The logging status indicates whether the database uses transaction logging and, if so, which log-buffering mechanism the database employs. To find out the transaction-logging status of a database, use the database server utilities, as explained in "Monitoring the Logging Mode of a Database" on page 12-6. The database-logging status indicates any of the following types of logging:

- Unbuffered transaction logging
- Buffered transaction logging
- ANSI-compliant transaction logging
- No logging

All logical-log records pass through the logical-log buffer in shared memory before the database server writes them to the logical log on disk. However, the point at which the database server flushes the logical-log buffer is different for buffered transaction logging and unbuffered transaction logging. For more information, see Figure 7-1 on page 7-2 and "Flushing the Logical-Log Buffer" on page 7-28.

Unbuffered Transaction Logging

If transactions are made against a database that uses unbuffered logging, the records in the logical-log buffer are guaranteed to be written to disk during commit processing. When control returns to the application after the COMMIT statement (and before the PREPARE statement for distributed transactions), the logical-log records are on the disk. The database server flushes the records as soon as any transaction in the buffer is committed (that is, a commit record is written to the logical-log buffer).

When the database server flushes the buffer, only the used pages are written to disk. Used pages include pages that are only partially full, however, so some space

is wasted. For this reason, the logical-log files on disk fill up faster than if all the databases on the same database server use buffered logging.

Unbuffered logging is the best choice for most databases because it guarantees that all committed transactions can be recovered. In the event of a failure, only uncommitted transactions at the time of the failure are lost. However, with unbuffered logging, the database server flushes the logical-log buffer to disk more frequently, and the buffer contains many more partially full pages, so it fills the logical log faster than buffered logging does.

Buffered Transaction Logging

If transactions are made against a database that uses buffered logging, the records are held (*buffered*) in the logical-log buffer for as long as possible. They are not flushed from the logical-log buffer in shared memory to the logical log on disk until one of the following situations occurs:

- The buffer is full.
- A commit on a database with unbuffered logging flushes the buffer.
- A checkpoint occurs.
- The connection is closed.

If you use buffered logging and a failure occurs, you cannot expect the database server to recover the transactions that were in the logical-log buffer when the failure occurred. Thus, you could lose some committed transactions. In return for this risk, performance during alterations improves slightly. Buffered logging is best for databases that are updated frequently (when the speed of updating is important), as long as you can re-create the updates in the event of failure. You can tune the size of the logical-log buffer to find an acceptable balance for your system between performance and the risk of losing transactions to system failure.

ANSI-Compliant Transaction Logging

The ANSI-compliant database logging status indicates that the database owner created this database using the MODE ANSI keywords. ANSI-compliant databases always use unbuffered transaction logging, enforcing the ANSI rules for transaction processing. You cannot change the buffering status of ANSI-compliant databases.

No Database Logging

If you turn off logging for a database, transactions are not logged, but other operations are logged. For more information, see "Activity That is Always Logged" on page 11-3. Usually, you would turn off logging for a database when you are loading data, or just executing queries.

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of database server processing. For example, if you are loading many rows into a database from a recoverable source such as tape or an ASCII file, you might not need transaction logging, and the loading would proceed faster without it. However, if other users are active in the database, you would not have logical-log records of their transactions until you reinitiate logging, which must wait for a level-0 backup.

Databases with Different Log-Buffering Status

All databases on a database server use the same logical log and the same logical-log buffers. Therefore, transactions against databases with different log-buffering statuses can write to the same logical-log buffer. In that case, if

transactions exist against databases with buffered logging *and* against databases with unbuffered logging, the database server flushes the buffer *either* when it is full *or* when transactions against the databases with unbuffered logging complete.

Database Logging in an X/Open DTP Environment

Databases in the X/Open distributed transaction processing (DTP) environment must use *unbuffered logging*. Unbuffered logging ensures that the database server logical logs are always in a consistent state and can be synchronized with the transaction manager. If a database created with buffered logging is opened in an X/Open DTP environment, the database status automatically changes to unbuffered logging. The database server supports both ANSI-compliant and non-ANSI databases. For more information, see "Transaction Managers" on page 25-1.

Settings or Changes for Logging Status or Mode

The user who creates a database with the CREATE DATABASE statement establishes the logging status or buffering mode for that database. For more information on the CREATE DATABASE statement, see the *IBM Informix Guide to SQL: Syntax*.

If the CREATE DATABASE statement does not specify a logging status, the database is created without logging.

Only the database server administrator can change logging status. Chapter 12, "Managing the Database-Logging Mode," on page 12-1, describes this topic. Ordinary end users cannot change database-logging status.

If a database does not use logging, you do not need to consider whether buffered or unbuffered logging is more appropriate. If you specify logging but do not specify the buffering mode for a database, the default is unbuffered logging.

End users *can* switch from unbuffered to buffered (but not ANSI-compliant) logging and from buffered to unbuffered logging for the *duration of a session*. The SET LOG statement performs this change within an application. For more information on the SET LOG statement, see the *IBM Informix Guide to SQL: Syntax*.

Chapter 12. Managing the Database-Logging Mode

In This Chapter

This chapter covers the following topics on changing the database-logging mode:

- Understanding database-logging mode
- Modifying database-logging mode with **ondblog**
- Modifying database-logging mode with **ontape**
- Modifying database-logging mode with ON–Monitor
- Monitoring transaction logging

As a database server administrator, you can alter the logging mode of a database as follows:

- Change transaction logging from buffered to unbuffered.
- Change transaction logging from unbuffered to buffered.
- Make a database ANSI compliant.
- Add transaction logging (buffered or unbuffered) to a database.
- End transaction logging for a database.

For information about database-logging mode, when to use transaction logging, and when to buffer transaction logging, see Chapter 11, "Logging," on page 11-1. To find out the current logging mode of a database, see "Monitoring the Logging Mode of a Database" on page 12-6.

For information on using SQL administration API commands instead of some **onblog** and **ontape** commands, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1 and the *IBM Informix Dynamic Server Administrator's Reference*.

Changing the Database-Logging Mode

You can use **ondblog**, **ontape**, or ISA to add or change logging. Then use ON–Bar, or **ontape** to back up the data. When you use ON–Bar or **ontape**, the database server must be in online, administration, or quiescent mode.

For information on ON–Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

Table 12-1 shows how the database server administrator can change the database-logging mode. Certain logging mode changes take place immediately, while other changes require a level-0 backup.

Table 12-1. Logging Mode Transitions

Converting from:	Converting to No Logging	Converting to Unbuffered Logging	Converting to Buffered Logging	Converting to ANSI Compliant
No logging	Not applicable	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)

Table 12-1. Logging Mode Transitions (continued)

Converting from:	Converting to No Logging	Converting to Unbuffered Logging	Converting to Buffered Logging	Converting to ANSI Compliant
Unbuffered logging	Yes	Not applicable	Yes	Yes
Buffered logging	Yes	Yes	Not applicable	Yes
ANSI compliant	Illegal	Illegal	Illegal	Not applicable

Some general points about changing the database-logging mode follow:

- While the logging status is being changed, the database server places an exclusive lock on the database to prevent other users from accessing the database, and frees the lock when the change is complete.
- If a failure occurs during a logging-mode change, check the logging mode in ISA or the flags in the **sysdatabases** table in the **sysmaster** database, after you restore the database server data. For more information, see "Monitoring the Logging Mode of a Database" on page 12-6. Then retry the logging-mode change.
- After you choose either buffered or unbuffered logging, an application can use the SQL statement SET LOG to change from one logging mode to the other. This change lasts for the duration of the session. For information on SET LOG, see the *IBM Informix Guide to SQL: Syntax*.
- If you add logging to a database, the change is not complete until the next level-0 backup of all the storage spaces for the database.

Modifying the Database-Logging Mode with ondblog

You can use the **ondblog** utility to change the logging mode for one or more databases. If you add logging to a database, you must create a level-0 backup of the dbspace(s) that contains the database before the change takes effect. For more information, see the section on using **ondblog** in the *IBM Informix Administrator's Reference*.

Changing the Buffering Mode with ondblog

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, execute the following command:

```
ondblog unbuf stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, execute the following command:

```
ondblog buf stores_demo
```

Canceling a Logging Mode Change with ondblog

To cancel the logging mode change request before the next level-0 backup occurs, execute the following command:

```
ondblog cancel stores_demo
```

You cannot cancel the logging changes that are executed immediately.

Ending Logging with ondblog

To end logging for two databases that are listed in a file called **dbfile**, execute the following command:

```
ondblog nolog -f dbfile
```

Making a Database ANSI Compliant with ondblog

To make a database called **stores_demo** into an ANSI-compliant database with **ondblog**, execute the following command:

```
ondblog ansi stores_demo
```

Changing the Logging Mode of an ANSI-Compliant Database

After you create or convert a database to ANSI mode, you cannot easily change it to any other logging mode. If you accidentally convert a database to ANSI mode, follow these steps to change the logging mode:

To change the logging mode

1. To unload the data, use **dbexport** or any other migration utility. The **dbexport** utility creates the **schema** file.

   For information about how to load and unload data, see the *IBM Informix Migration Guide*.
2. To re-create a database with buffered logging and load the data, use the **dbimport -l buffered** command.

   To re-create a database with unbuffered logging and load the data, use the **dbimport -l** command.

Modifying the Database Logging Mode with ontape

If you use **ontape** as your backup tool, you can use **ontape** to change the logging mode of a database.

Turning On Transaction Logging with ontape

Before you modify the database-logging mode, read "Changing the Database-Logging Mode" on page 12-1.

You add logging to a database with **ontape** at the same time that you create a level-0 backup.

For example, to add buffered logging to a database called **stores_demo** with **ontape**, execute the following command:

```
ontape -s -B stores_demo
```

To add unbuffered logging to a database called **stores_demo** with **ontape**, execute the following command:

```
ontape -s -U stores_demo
```

In addition to turning on transaction logging, these commands create full-system storage-space backups. When **ontape** prompts you for a backup level, specify a level-0 backup.

Tip: With **ontape**, you must perform a level-0 backup of all storage spaces.

Ending Logging with ontape

To end logging for a database called **stores_demo** with **ontape**, execute the following command:

```
ontape -N stores_demo
```

Changing Buffering Mode with ontape

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, execute the following command:

```
ontape -U stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, execute the following command:

```
ontape -B stores_demo
```

Making a Database ANSI Compliant with ontape

To make a database called **stores_demo**, which already uses transaction logging (either unbuffered or buffered), into an ANSI-compliant database with **ontape**, execute the following command:

```
ontape -A stores_demo
```

To make a database called **stores_demo**, which does not already use transaction logging, into an ANSI-compliant database with **ontape**, execute the following command:

```
ontape -s -A stores_demo
```

In addition to making a database ANSI compliant, this command also creates a storage-space backup at the same time. Specify a level-0 backup when you are prompted for a level.

Tip: After you change the logging mode to *ANSI* compliant, you cannot easily change it again. To change the logging mode of *ANSI*-compliant databases, unload the data, re-create the database with the new logging mode, and reload the data. For details, see "Changing the Logging Mode of an ANSI-Compliant Database" on page 12-3.

Modifying Database Logging Mode with ISA

ISA uses the **ondblog** utility to modify the database-logging mode. If you turn on logging, perform a level-0 backup. For more information, see the ISA online help and "Modifying the Database-Logging Mode with ondblog" on page 12-2.

Modifying Database Logging Mode with ON-Monitor (UNIX)

You can use ON–Monitor to change the log-buffering mode between unbuffered and buffered. If you want to add logging to a database or make it ANSI compliant, you cannot use ON–Monitor. You must use **ontape**.

To change the log-buffering mode for a database, select the **Logical-Logs > Databases** option.

Modifying the Table-Logging Mode

The database server creates standard tables that use logging by default. To create a nonlogging table, use the CREATE TABLE statement with the WITH LOG clause. For information on the CREATE TABLE and ALTER TABLE statements, see the *IBM Informix Guide to SQL: Syntax*. For more information, refer to "Table Types for Dynamic Server" on page 9-23.

Altering a Table to Turn Off Logging

To switch a table from logging to nonlogging, use the SQL statement ALTER TABLE with the TYPE option of RAW. For example, the following statement changes table **tablog** to a RAW table:

```
ALTER TABLE tablog TYPE (RAW)
```

Altering a Table to Turn On Logging

To switch from a nonlogging table to a logging table, use the SQL statement ALTER TABLE with the TYPE option of STANDARD. For example, the following statement changes table **tabnolog** to a STANDARD table:

```
ALTER TABLE tabnolog TYPE (STANDARD)
```

Warning: When you alter a table to STANDARD, you turn logging on for that table. After you alter the table, perform a level-0 backup if you need to be able to restore the table.

Disabling Logging on Temporary Tables

You can disable logging on temporary tables to improve performance and to prevent Dynamic Server from transferring temporary tables when using a primary server in a data replication environment such as with HDR secondary, RS secondary, and SD secondary servers.

To disable logging on temporary tables, set the TEMPTAB_NOLOG configuration parameter to 1.

You can use the **onmode -wf** command to change the value of TEMPTAB_NOLOG.

Monitoring Transactions

This section contains references for information on ways to monitor transactions.

Command	Description	Reference
onstat -x	Monitor transactions.	"Monitoring a Global Transaction" on page 25-14
onstat -g sql	Monitor SQL statements, listed by session ID and database.	Performance monitoring in the *IBM Informix Dynamic Server Performance Guide*

Command	Description	Reference
onstat -g stm	Monitor memory usage of prepared SQL statements.	Memory utilization in the *IBM Informix Dynamic Server Performance Guide*

Monitoring the Logging Mode of a Database

This section discusses ways to monitor the logging mode of your database and tables.

Monitoring the Logging Mode with SMI Tables

Query the **sysdatabases** table in the **sysmaster** database to determine the logging mode. This table contains a row for each database that the database server manages. The **flags** field indicates the logging mode of the database. The **is_logging**, **is_buff_log**, and **is_ansi** fields indicate whether logging is active, and whether buffered logging or ANSI-compliant logging is used. For a description of the columns in this table, see the **sysdatabases** section in the chapter about the **sysmaster** database in the *IBM Informix Administrator's Reference*.

Monitoring the Logging Mode with ON-Monitor (UNIX)

To use ON–Monitor to find out the logging mode of a database, select the **Status > Databases** option. When database logging is on, ON–Monitor shows the buffering mode. ON–Monitor can only display up to 100 databases. If you have more than 100 databases on your database server, use the SMI tables to display the full list, as described in "Monitoring the Logging Mode with SMI Tables."

Monitoring the Logging Mode with ISA

To use ISA to display the logging status and buffering mode for your databases, select **SQL > Schema**.

Chapter 13. Logical Log

In This Chapter

The information in Chapter 11, "Logging," on page 11-1, and this chapter will help you understand how the database server uses the logical log. For information on how to perform logical-log tasks, see Chapter 14, "Managing Logical-Log Files," on page 14-1, and Chapter 12, "Managing the Database-Logging Mode," on page 12-1.

What Is the Logical Log?

To keep a history of transactions and database server changes since the time of the last storage-space backup, the database server generates log records. The database server stores the log records in the *logical log*, a circular file that is composed of three or more logical-log files. The log is called *logical* because the log records represent logical operations of the database server, as opposed to physical operations. At any time, the combination of a storage-space backup plus logical-log backup contains a complete copy of your database server data.

As the database server administrator, you must configure and manage the logical log. For example, if you do not back up the log files regularly, the logical log fills and the database server suspends processing.

These responsibilities include the following tasks:

- Choosing an appropriate location for the logical log

  See "Location of Logical-Log Files."
- Monitoring the logical-log file status

  See "Identification of Logical-Log Files" on page 13-2.
- Allocating an appropriate amount of disk space for the logical log

  See "Size of the Logical Log" on page 13-3.
- Allocating additional log files whenever necessary

  See "Allocating Log Files" on page 14-9.
- Backing up the logical-log files to media

  See "Backing Up Logical-Log Files" on page 14-4 and "Freeing of Logical-Log Files" on page 13-5.
- Managing logging of blobspaces and sbspaces

  See "Logging Blobspaces and Simple Large Objects" on page 13-6 and "Logging Sbspaces and Smart Large Objects" on page 13-7.

Location of Logical-Log Files

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace. You have no control over this action. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize contention), move the logical-log files out of the root dbspace to a dbspace on a disk that is not shared by active tables or the physical log. See "Moving a Logical-Log File to Another Dbspace" on page 14-13.

To improve performance further, separate the logical-log files into two groups and store them on two separate disks (neither of which contains data). For example, if you have six logical-log files, you might locate files 1, 3, and 5 on disk 1, and files 2, 4, and 6 on disk 2. This arrangement improves performance because the same disk drive never has to handle writes to the current logical-log file and backups to tape at the same time.

The logical-log files contain critical information and should be mirrored for maximum data protection. If you move logical-log files to a different dbspace, plan to start mirroring on that dbspace.

Identification of Logical-Log Files

Each logical-log file, whether backed up to media or not, has a *unique* ID number. The sequence begins with 1 for the first logical-log file filled after you initialize the database server disk space. When the current logical-log file becomes full, the database server switches to the next logical-log file and increments the unique ID number for the new log file by one. Log files that are newly added or marked for deletion have unique ID numbers of `0`.

The actual disk space allocated for each logical-log file has an identification number known as the *log file number*. For example, if you configure six logical-log files, these files have log numbers one through six. The log numbers might be out of sequence. As logical-log files are backed up and freed, the database server reuses the disk space for the logical-log files.

Table 13-1 illustrates the relationship between the log numbers and the unique ID numbers. Log 7 is inserted after log 5 and used for the first time in the second rotation.

Table 13-1. Logical-Log File-Numbering Sequence

Log File Number	First Rotation Unique ID Number	Second Rotation Unique ID Number	Third Rotation Unique ID Number
1	1	7	14
2	2	8	15
3	3	9	16
4	4	10	17
5	5	11	18
7	**0**	12	19
6	6	13	20

Status Flags of Logical-Log Files

All logical-log files have one of the following status flags in the first position: Added (**A**), Deleted (**D**), Free (**F**), or Used (**U**). Table 13-2 shows the possible log-status flag combinations.

Table 13-2. Logical-Log Status Flags

Status Flag	Status of Logical-Log File
A------	Log file has been *added*, and is available, but has not yet been used.

Table 13-2. Logical-Log Status Flags (continued)

Status Flag	Status of Logical-Log File
D------	If you drop a log file with a status of U-B, it is marked as *deleted*. This log file is dropped and its space is freed for reuse when you take a level-0 backup of all storage spaces.
F------	Log file is *free* and available for use. A logical-log file is freed after it is backed up, all transactions within the logical-log file are closed, and the oldest update stored in this file is flushed to disk.
U	Log file has been *used* but not backed up.
U-B----	Log file is *backed up* but still needed for recovery. (The log file is freed when it is no longer needed for recovery.)
U-B---L	Log is backed up but still needed for recovery. Contains the last checkpoint record.
U---C	The database server is *currently* filling the log file.
U---C-L	This current log file contains the *last* checkpoint record.

Use the **onstat -l** command to list the log files by number and monitor the status flags and percentage of log space used. For more details, see "onstat -l" on page 14-7.

Size of the Logical Log

Decide how many log files you want and what size. If you allocate more disk space than necessary, space is wasted. If you do not allocate enough disk space, however, performance might be adversely affected. Consider these points about the size and number of the logical-log files:

- The minimum size for a logical-log file is 200 kilobytes.
- The maximum size for a logical-log file is 1048576 pages (equivalent to 0x100000).
- Smaller log files mean that you can recover to a later time if the disk that contains the log files goes down. If continuous log backup is set, log files are automatically backed up as they fill. Smaller logs result in slightly longer logical recovery.
- Use larger log files when many users are writing to the logs at the same time.

Number of Logical-Log Files

When you think about the number of logical-log files, consider these points:

- You must always have at least three logical-log files and a maximum of 32,767 log files. The number of log files depends on the size of the log files.
- The number of log files affects the frequency of logical-log backups.
- The number of logical-log files affects the rate at which blobspace blobpages can be reclaimed. See "Backing Up Log Files to Free Blobpages" on page 13-6.

Performance Considerations

For a given level of system activity, the less logical-log disk space that you allocate, the sooner that logical-log space fills up, and the greater the likelihood that user activity is blocked due to backups and checkpoints. Tune the logical-log size to find the optimum value for your system.

- Logical-log backups

  When the logical-log files fill, you have to back them up. The backup process can hinder transaction processing that involves data located on the same disk as the logical-log files. Put the physical log, logical logs, and user data on separate disks. (See the *IBM Informix Backup and Restore Guide*.)

- Size of the logical log

  A smaller logical log fills faster than a larger logical log. You can add a larger logical-log file, as explained in "Adding Logical-Log Files Manually" on page 14-11.

- Size of individual logical-log records

  The sizes of the logical-log records vary, depending on both the processing operation and the database server environment. In general, the longer the data rows, the larger the logical-log records. The logical log contains images of rows that have been inserted, updated, or deleted. Updates can use up to twice as much space as inserts and deletes because they might contain both before-images and after-images. (Inserts store only the after-image and deletes store only the before-image.)

- Number of logical-log records

  The more logical-log records written to the logical log, the faster it fills. Databases with transaction logging fill the logical log faster than transactions against databases without transaction logging.

- Type of log buffering

  Databases that use unbuffered transaction logging fill the logical log faster than databases that use buffered transaction logging.

- Enterprise Replication on a table

  Because Enterprise Replication generates before-images and after-images of the replicated tables, it could cause the logical log to fill.

- Frequency of rollbacks

  More rollbacks fill the logical log faster. The rollbacks themselves require logical-log file space although the rollback records are small.

- Number of smart large objects

  Smart large objects that have user data logging enabled and have a large volume of user data updates can fill logical logs at a prodigious rate. Use temporary smart large objects if you do not want to log the metadata.

Dynamic Log Allocation

Dynamic log allocation prevents log files from filling and hanging the system during long transaction rollbacks. The only time that this feature becomes active is when the next log file contains an open transaction. (A *transaction* is *long* if it is not committed or rolled back when it reaches the long-transaction high-watermark.)

The database server automatically (dynamically) allocates a log file after the current log file when the next log file contains an open transaction. Dynamic log allocation allows you to do the following actions:

- Add a log file while the system is active

- Insert a log file after the current log file
- Immediately access new log files even if the root dbspace is not backed up

The best way to test dynamic log allocation is to produce a transaction that spans all the log files and then use **onstat -l** to check for newly added log files. For more information, see "Allocating Log Files" on page 14-9.

Important: You still must back up log files to prevent them from filling. If the log files fill, the system hangs until you perform a backup.

Freeing of Logical-Log Files

Each time the database server commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began. The following criteria must be satisfied before the database server frees a logical-log file for reuse:

- The log file is backed up.
- No records within the logical-log file are associated with open transactions.
- The logical-log file does not contain the oldest update not yet flushed to disk.

Action If the Next Logical-Log File Is Not Free

If the database server attempts to switch to the next logical-log file but finds that the next log file in sequence is still in use, the database server immediately suspends all processing. Even if other logical-log files are free, the database server cannot skip a file in use and write to a free file out of sequence. Processing stops to protect the data within the logical-log file.

The logical-log file might be in use for any of the following reasons:

- The file contains the latest checkpoint or the oldest update not yet flushed to disk.

  Issue the **onmode -c** command to perform a checkpoint and free the logical-log file. For more information, see "Forcing a Checkpoint" on page 16-4.
- The file contains an open transaction.

  The open transaction is the long transaction discussed in "Setting High-Watermarks for Rolling Back Long Transactions" on page 14-16.
- The file is not backed up.

  If the logical-log file is not backed up, processing resumes when you use ON–Bar or **ontape** to back up the logical-log files.

Action if the Next Log File Contains the Last Checkpoint

The database server does not suspend processing when the next log file contains the last checkpoint or the oldest update. The database server always forces a checkpoint when it switches to the last available log, if the previous checkpoint record or oldest updated not-yet-flushed-to-disk is located in the log that follows the last available log. For example, if four logical-log files have the status shown in the following list, the database server forces a checkpoint when it switches to logical-log file 3.

Log File Number
Logical-Log File Status

1 U-B----

2 U---C--

3 F

4 U-B---L

Logging Blobspaces and Simple Large Objects

Simple-large-object data (TEXT and BYTE data types) is potentially too voluminous to include in a logical-log record. If simple large objects are always logged, they might be so large that they slow down the logical log.

The database server assumes that you designed your databases so that smaller simple large objects are stored in dbspaces and larger simple large objects are stored in blobspaces:

- The database server includes simple-large-object data in log records for simple large objects stored in dbspaces.
- The database server does not include simple-large-object data in log records for simple large objects stored in blobspaces. The logical log records blobspace data only when you back up the logical logs.

To obtain better overall performance for applications that perform frequent updates of simple large objects in blobspaces, reduce the size of the logical log. Smaller logs can improve access to simple large objects that must be reused. For more information, see the chapter on configuration effects on I/O utilization in your *IBM Informix Dynamic Server Performance Guide*.

Switching Log Files to Activate Blobspaces

You must switch to the next logical-log file in these situations:

- After you create a blobspace, if you intend to insert simple large objects in the blobspace right away
- After you add a new chunk to an existing blobspace, if you intend to insert simple large objects in the blobspace that will use the new chunk

The database server requires that the statement that creates a blobspace, the statement that creates a chunk in the blobspace, and the statements that insert simple large objects into that blobspace appear in separate logical-log files. This requirement is independent of the database-logging status.

For instructions on switching to the next log file, see "Switching to the Next Logical-Log File" on page 14-5.

Backing Up Log Files to Free Blobpages

When you delete data stored in blobspace pages, those pages are not necessarily freed for reuse. The blobspace pages are free only when *both* of the following actions have occurred:

- The TEXT or BYTE data has been deleted, either through an UPDATE to the column or by deleting the row
- The logical log that stores the INSERT of the row that has TEXT or BYTE data is backed up

Backing Up Blobspaces After Inserting or Deleting TEXT and BYTE Data

Be sure to back up all blobspaces and logical logs containing transactions on simple large objects stored in a blobspace. During log backup, the database server uses the data pointer in the logical log to copy the changed text and byte data from the blobspace into the logical log.

Logging Sbspaces and Smart Large Objects

Sbspaces, described in "Sbspaces" on page 9-12, contain two components: metadata and user data. By default, sbspaces are *not* logged.

The *metadata* component of the sbspace describes critical characteristics of smart large objects stored in a particular sbspace. The metadata contains pointers to the smart large objects. If the metadata were to be damaged or become inaccessible, the sbspace would be corrupted and the smart large objects within that sbspace would be unrecoverable.

Metadata in a standard sbspace is *always* logged, even if logging is turned off for a database. Logging sbspace metadata ensures that the metadata can always be recovered to a consistent transaction state. However, metadata in a temporary sbspace is *not* logged.

Using Sbspace Logging

When an sbspace is logged, the database server slows down, and the logical logs fill up quickly. If you use logging for sbspaces, you must ensure that the logical logs are large enough to hold the logging data. For more information, see "Estimating the Log Size When Logging Smart Large Objects" on page 14-3.

When you turn on logging for a database, the database server does not begin logging until you perform a level-0 backup. However, when you turn on logging for a smart large object, the database server begins logging changes to it immediately. To reduce the volume of log entries, you could load smart large objects with logging turned off and then turn logging back on to capture updates to the smart large objects.

Warning: When you turn logging on for a smart large object, you must immediately perform a level-0 backup to be able to recover and restore the smart large object.

For more information, see "Backing Up Sbspaces" on page 14-4 and the *IBM Informix Backup and Restore Guide*.

Choosing Logging for Smart Large Objects

Use logging for smart large objects if users are updating the data frequently or if the ability to recover any updated data is critical. The database server writes a record of the operation (insert, update, delete, read, or write) to the logical-log buffer. The modified portion of the CLOB or BLOB data is included in the log record.

To increase performance, turn off logging for smart large objects. Also turn off logging if users are primarily analyzing the data and updating it infrequently, or if the data is not critical to recover.

Logging for Updated Smart Large Objects

When you update a smart large object, the database server does not log the entire object. Assume that the user is writing X bytes of data at offset Y with logging enabled for smart large objects. The database server logs the following:

- If Y is set to the end of the large object, the database server logs X bytes (the updated byte range).
- If Y is at the beginning or in the middle of the large object, the database server logs the smallest of these choices:
  - Difference between the old and new image
  - Before-image and after-image
  - Nothing is logged if the before- and after-images are the same

Turning Logging On or Off for an Sbspace

If you want to use logging in an sbspace, specify the **-Df LOGGING=ON** option of the **onspaces** command when you create the sbspace. If logging is turned off in the sbspace, you can turn on logging for smart large objects in specific columns. One column that contains smart large objects could have logging turned on while another column has logging turned off.

To verify that smart large objects in an sbspace are logged, use the following command:

```
oncheck -pS sbspace | grep "Create Flags"
```

If you create smart large objects in the sbspace with the default logging option and you see the LO_NOLOG flag in the output, the smart large objects in this sbspace are not logged. If you see the LO_LOG flag in the output, all smart large objects in this sbspace are logged.

You can modify the logging status of an sbspace in any of the following ways.

Function or Statement to Specify	Logging Action	References
onspaces -ch -Df LOGGING=ON	Turns logging on or off for an existing sbspace	"Altering Storage Characteristics of Smart Large Objects" on page 10-25 *IBM Informix Administrator's Reference*
LOG option in the PUT clause of the CREATE TABLE or alter table statement	Turns on logging for all smart large objects that you load into the column	"Logging" on page 9-16 *IBM Informix Guide to SQL: Syntax*
mi_lo_create DataBlade API function	Turns off logging for a smart large object when it is initially loaded	*IBM Informix DataBlade API Function Reference*
mi_lo_alter DataBlade API function	Turns on logging after the load is complete	*IBM Informix DataBlade API Function Reference*
ifx_lo_create Informix ESQL/C function	Turns off logging for a smart large object when it is initially loaded	*IBM Informix ESQL/C Programmer's Manual*
ifx_lo_alter Informix ESQL/C function	Turns on logging after the load is complete	*IBM Informix ESQL/C Programmer's Manual*

Using Smart-Large-Object Log Records

When you create a smart large object with the LOG option, the logical log creates a *smart-blob log record*. Smart-blob log records track changes to user data or metadata. When smart large objects are updated, only the modified portion of the sbpage appears in the log record. User-data log records appear in the logical log only when logging is enabled for the smart large object.

Warning: Be careful about enabling logging for smart large objects that are updated frequently. This logging overhead might significantly slow down the database server.

For information on the log records for smart large objects, see the chapter on interpreting logical-log records in the *IBM Informix Dynamic Server Administrator's Reference*.

Preventing Long Transactions When Logging Smart-Large-Object Data

You can use smart large objects in situations where the data collection process for a single smart large object lasts for long periods of time. Consider, for example, an application that records many hours of low-quality audio information. Although the amount of data collected might be modest, the recording session might be long, resulting in a long-transaction condition.

Tip: To prevent long transactions from occurring, periodically commit writes to smart large objects.

Logging Process

This section describes in detail the logging process for dbspaces, blobspaces, and sbspaces. This information is not required for performing normal database server administration tasks.

Dbspace Logging

The database server uses the following logging process for operations that involve data stored in dbspaces:

1. Reads the data page from disk to the shared-memory page buffer
2. Copies the unchanged page to the physical-log buffer, if needed
3. Writes the new data to the page buffer and creates a logical-log record of the transaction, if needed
4. Flushes the physical-log buffer to the physical log on disk
5. Flushes the logical-log buffer to a logical-log file on disk
6. Flushes the page buffer and writes it back to disk

Blobspace Logging

The database server logs blobspace data, but the data does not pass through either shared memory or the logical-log files on disk. The database server copies data

stored in a blobspace directly from disk to tape. Records of modifications to the blobspace overhead pages (the free-map and bitmap pages) are the only blobspace data that reaches the logical log.

Chapter 14. Managing Logical-Log Files

In This Chapter

You must manage logical-log files even if none of your databases uses transaction logging. See Chapter 13, "Logical Log," on page 13-1 for background information on logical logs.

You must log in as either **informix** or **root** on UNIX to make any of the changes described in this chapter. You must be a member of the **Informix-Admin** group on Windows.

You perform these tasks when setting up your logical log:

- Before you initialize or restart the database server, use the LOGFILES parameter to specify the number of logical-log files to create.
- After the database server is online, estimate the size and number of logical-log files that your system will need.

  See "Estimating the Size and Number of Log Files."
- If you do not want to use the default values, change the LOGSIZE and LOGBUFF configuration parameters.

  See "Changing Logging Configuration Parameters" on page 14-14.
- Add the estimated number of logical-log files.

  See "Allocating Log Files" on page 14-9.

You perform the following tasks routinely:

- Backing up a logical-log file
- Switching to the next logical-log file
- Freeing a logical-log file
- Monitoring logging activity and log-backup status

You perform these tasks occasionally, if necessary:

- Adding a logical-log file
- Dropping a logical-log file
- Changing the size of a logical-log file
- Moving a logical-log file
- Changing the logical-log configuration parameters
- Monitoring event alarms for logical logs
- Setting high-watermarks for transactions

For information on using SQL administration API commands instead of some **oncheck**, **onmode**, **onparams** and **onspaces** commands, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1 and the *IBM Informix Dynamic Server Administrator's Reference*.

Estimating the Size and Number of Log Files

Use the LOGSIZE configuration parameter to set the size of the logical-log files.

The amount of log space that is optimal for your database server system depends on the following factors:

- Your application requirements and the amount of update activity your applications experience. Increased update activity requires increased log space.
- The recovery time objective (RTO) standards for the amount of time, in seconds, that the server has to recover from a problem after you restart the server and bring it into online or quiescent mode.

  In the case of a catastrophic event, consider how much data loss you can tolerate. More frequent log backups, which reduce the risk of data and transaction loss, require increased log space.
- Whether you use Enterprise Replication or data replication configurations such as HDR secondary, SD secondary or RS secondary servers.

  These replication services can influence the number and size of log files. If your system uses any of these replication services, see guidelines in Chapter 20, "Using High-Availability Data Replication (Enterprise/Workgroup Editions)," on page 20-1 or in the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Some guidelines for determining log size are:

- Generally, you can more easily manage a few large log files than you can manage many small log files.
- Too much log space does not impact performance. However, not having enough log files and log space can impact performance, because the database server triggers frequent checkpoints.
- Smart large objects in blobspaces are not logged, but they are included in the log backup in which the object was created. This means that the objects are not freed until the server backs up the log in which they were created. Therefore, if smart large objects in a blobspace are frequently updated, you might need more frequent log backups to acquire additional free space within a blobspace.
- For applications that generate a small amount of log data, start with 10 log files of 10 megabytes each.
- For applications that generate a large amount of log data, start with start with 10 log files with 100 megabytes.

There are two ways to maintain an RPO policy, which determines the tolerance for loss of data in case of a catastrophic event such as the loss of the data server:

- One way to maintain an RPO policy is to use automatic log backups that trigger log backups whenever a log file fills up. This limits data loss to the transactions contained in the log file during the backup, plus any additional transactions that occur during the log backup.
- Another way to maintain an RPO policy is to use the Scheduler. You can create a task that automatically backs up any new log data at timed intervals since the last log backup. This limits data loss to the transactions not backed up between time intervals. For information on using the Scheduler, see Chapter 30, "The Scheduler," on page 30-1.

If an RPO policy is required, you can use the Scheduler to insert a task that executes at the desired frequency to maintain the policy. This automatically backs up log files at certain times within the daily cycle. Should the log space fill prior to the logs being backed up and recycled, you can back up the logs and add a new log file to allow transaction processing to continue, or you can use the Scheduler to add a new task to detect this situation and perform either operation automatically.

You can add log files at any time, and the database server automatically adds log files when required for transaction consistency, for example, for long transactions that might consume large amounts of log space.

The easiest way to increase the amount of space for the logical log is to add another logical-log file. See "Adding Logical-Log Files Manually" on page 14-11.

The following expression provides an example total-log-space configuration, in kilobytes:

```
LOGSIZE = (((connections * maxrows) * rowsize) / 1024) / LOGFILES
```

Expression Element	Explanation
LOGSIZE	Specifies the size of each logical-log file in kilobytes.
connections	Specifies the maximum number of connections for all network types that you specify in the **sqlhosts** file or registry and in the NETTYPE parameter. If you configured more than one connection by setting multiple NETTYPE configuration parameters in your configuration file, add the users fields for each NETTYPE, and substitute this total for *connections* in the preceding formula.
maxrows	Specifies the largest number of rows to be updated in a single transaction.
rowsize	Specifies the average size of a table row in bytes. To calculate the *rowsize*, add the length (from the **syscolumns** system catalog table) of the columns in the row.
1024	Converts the LOGSIZE to the specified units of kilobytes.
LOGFILES	Specifies the number of logical-log files.

Estimating the Log Size When Logging Smart Large Objects

If you plan to log smart-large-object user data, you must ensure that the log size is *considerably* larger than the amount of data being written. If you store smart large objects in standard sbspaces, the metadata is always logged, even if the smart large objects are not logged. If you store smart large objects in temporary sbspaces, there is no logging at all.

Estimating the Number of Logical-Log Files

The LOGFILES parameter provides the number of logical-log files at system initialization or restart. If all your logical-log files are the same size, you can calculate the total space allocated to the logical-log files as follows:

```
total logical log space = LOGFILES * LOGSIZE
```

If the database server contains log files of different sizes, you cannot use the (LOGFILES * LOGSIZE) expression to calculate the size of the logical log. Instead, you need to add the sizes for each individual log file on disk. Check the **size** field in the **onstat -l** output. For more information, see "onstat -l" on page 14-7.

For information on LOGSIZE, LOGFILES, and NETTYPE, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

Backing Up Logical-Log Files

The logical logs contain a history of the transactions that have been performed. The process of copying a logical-log file to media is referred to as *backing up* a logical-log file. Backing up logical-log files achieves the following two objectives:

- It stores the logical-log records on media so that they can be rolled forward if a data restore is needed.
- It makes logical-log-file space available for new logical-log records.

  If you neglect to back up the log files, you can run out of log space.

You can initiate a manual logical-log backup or set up continuous logical-log backups. After you restore the storage spaces, you must restore the logical logs to bring the data to a consistent state. For more information on log backups, see the *IBM Informix Backup and Restore Guide*.

Backing Up Blobspaces

It does not matter whether you back up the logical logs or blobspaces first.

To back up blobspace data

1. Close the current logical log if it contains transactions on simple large objects in a blobspace.
2. Perform a backup of the logical logs and blobspace as soon as possible after updating simple-large-object data.

Warning: If you do not back up these blobspaces and logical logs, you might not be able to restore the blobspace data. If you wait until a blobspace is down to perform the log backup, the database server cannot access the blobspace to copy the changed data into the logical log.

Backing Up Sbspaces

When you turn on logging for smart large objects, you must perform a level-0 backup of the sbspace.

Figure 14-1 on page 14-5 shows what happens if you turn on logging in an sbspace that is not backed up. The unlogged changes to smart large object **LO1** are lost during the failure, although the logged changes are recoverable. You will not be able to fully restore **LO1**.

During fast recovery, the database server rolls forward all committed transactions for **LO1**. If **LO1** is unlogged, the database server would be unable to roll back uncommitted transactions. Then the **LO1** contents would be incorrect. For more information, refer to “Fast Recovery” on page 15-7.

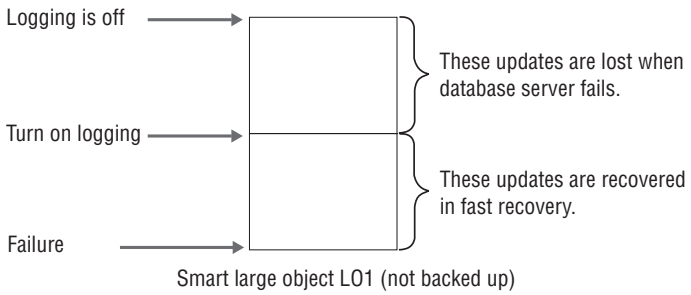


Figure 14-1. Turning On Logging in an Sbspace

Switching to the Next Logical-Log File

You might want to switch to the next logical-log file before the current log file becomes full for the following reasons:

- To back up the current log
- To activate new blobspaces and blobspace chunks

The database server can be in online mode to make this change. Execute the following command to switch to the next available log file:

```
onmode -l
```

The change takes effect immediately. (Be sure that you type a lowercase L on the command line, not a number 1.)

Freeing a Logical-Log File

If a log file is newly added (status **A**), it is immediately available for use. It also can be dropped immediately.

You might want to free a logical-log file for the following reasons:

- So that the database server does not stop processing
- To free the space used by deleted blobpages

The procedures for freeing log files vary, depending on the status of the log file. Each procedure is described in the following sections. To find out the status of logical-log files, see "Status Flags of Logical-Log Files" on page 13-2 and "Monitoring Logging Activity" on page 14-7.

Tip: For information using ON–Bar or **ontape** to back up storage spaces and logical logs, refer to the *IBM Informix Backup and Restore Guide*.

Deleting a Log File with Status D

When you drop a used log file, it is marked as deleted (status **D**) and cannot be used again, and **onparams** prints this message:

```
Log file log_file_number has been pre-dropped. It will be
deleted from the log list and its space can be reused
once you take level 0 archives of all BLOBspaces,
Smart BLOBspaces and non-temporary DBspaces.
```

The level 0 archive is necessary to make sure that the log file itself and all of the associated information in the different dbspaces has been archived. The log file is deleted at the end of the level 0 archive; however, because the removal of the log file is itself a change in the root reserved pages structure on the disk, the next archive to be taken also needs to be a level 0 archive. The level 0 archive must occur before a level 1 or level 2 archive can be performed.

Freeing a Log File with Status U

If a log file contains records, but is not yet backed up (status **U**), back up the file using the backup tool that you usually use.

If backing up the log file does not change the status to free (**F**), its status changes to either **U-B** or **U-B-L**. See "Freeing a Log File with Status U-B or F," following, or "Freeing a Log File with Status U-B-L" on page 14-7.

Freeing a Log File with Status U-B or F

If a log file is backed up but still in use (status **U-B**), some transactions in the log file are still under way, or the log file contains the oldest update that is required for fast recovery. Because a log file with status **F** has been used in the past, it follows the same rules as for status **U-B**.

To free a backed up log file that is in use

1. If you do not want to wait until the transactions complete, take the database server to quiescent mode. See "Change Immediately from Online to Quiescent Mode" on page 4-13. Any active transactions are rolled back.
2. Use the **onmode -c** command to force a checkpoint. Do this because a log file with status **U-B** might contain the oldest update.

A log file that is backed up but *not* in use (status **U-B**) does not need to be freed. In the following example, log 34 does not need to be freed, but logs 35 and 36 do. Log 35 contains the last checkpoint, and log 36 is backed up but still in use.

```
34 U-B--   Log is used, backed up, and not in use
35 U-B-L   Log is used, backed up, contains last checkpoint
36 U-B--   Log is used, backed up, and not in use
37 U-C--   This is the current log file, not backed up
```

Tip: You can free a logical log with a status of U-B (and not L) only if it is not spanned by an active transaction and does not contain the oldest update.

Freeing a Log File with Status U-C or U-C-L

Follow these steps to free the current log file.

To free the current log file (status C)

1. Execute the following command to switch the current log file to the next available log file:

   ```
   onmode -l
   ```

2. Back up the original log file with ON–Bar or **ontape**.
3. After all full log files are backed up, you are prompted to switch to the next available log file and back up the new current log file.

   You do not need to do the backup because you just switched to this log file.

After you free the current log file, if the log file has status **U-B** or **U-B-L**, refer to "Freeing a Log File with Status U-B or F" on page 14-6 or "Freeing a Log File with Status U-B-L."

Freeing a Log File with Status U-B-L

If a log file is backed up and all transactions within it are closed but the file is not free (status **U-B-L**), this logical-log file contains the most-recent checkpoint record.

To free log files with a status **U-B-L**, the database server must create a new checkpoint. You can execute the following command to force a checkpoint:

```
onmode -c
```

UNIX Only:

To force a checkpoint with ON–Monitor, select the **Force-Ckpt** option.

Monitoring Logging Activity

Monitor the logical-log files to determine the total available space (in all the files), the space available in the current file, and the status of a file (for example, whether the log has been backed up yet). For information on monitoring the logical-log buffers, see "Monitoring Physical and Logical-Logging Activity" on page 16-2.

Monitoring the Logical Log for Fullness

You can use the following command-line utilities to monitor logical-log files.

onstat -l

The **onstat -l** command displays information about physical and logical logs.

The output section that contains information on each logical-log file includes the following information:

- The address of the logical-log file descriptor
- The log file number
- Status flags that indicate the status of each log (free, backed up, current, and so on)
- The unique ID of the log file
- The beginning page of the file
- The size of the file in pages, the number of pages used, and the percentage of pages used

The log file numbers in the **numbers** field can get out of sequence if you drop several logs in the middle of the list or if the database server dynamically adds log files.

For more information on and an example of **onstat -l** output, see the *IBM Informix Administrator's Reference*.

oncheck -pr

The database server stores logical-log file information in the reserved pages dedicated to checkpoint information. Because the database server updates this

information only during a checkpoint, it is not as recent as the information that the **onstat -l** option displays. For more details on using these options to display reserved page information, see the *IBM Informix Dynamic Server Administrator's Reference*.

You can view the checkpoint reserved pages with the **oncheck -pr** command. Figure 14-2 shows sample output for one of the logical-log files.

```
...
Log file number             1
Unique identifier           7
Log contains last checkpoint Page 0, byte 272
Log file flags              0x3  Log file in use
                                 Current log file
Physical location           0x1004ef
Log size                    750 (p)
Number pages used           1
Date/Time file filled       01/29/2001 14:48:32
...
```

Figure 14-2. oncheck -pr Output Containing Logical-Log File Information

Monitoring Temporary Logical Logs

The database server uses *temporary logical logs* to roll forward transactions during a warm restore, because the permanent logs are not available then. When the roll forward completes, the database server frees the temporary log files. If you issue **onstat -l** during a warm restore, the output includes a fourth section on temporary log files in the same format as regular log files. Temporary log files use only the **B**, **C**, **F**, and **U** status flags.

Using SMI Tables

Query the **syslogs** table to obtain information on logical-log files. This table contains a row for each logical-log file. The columns are as follows.

Column
Description

number
Identification number of the logical-log file

uniqid
Unique ID of the log file

size Size of the file in pages

used Number of pages used

is_used
Flag that indicates whether the log file is being used

is_current
Flag that indicates whether the log file is current

is_backed_up
Flag that indicates whether the log file has been backed up

is_new
Flag that indicates whether the log file has been added since the last storage space backup

is_archived
Flag that indicates whether the log file has been written to the archive tape

is_temp
Flag that indicates whether the log file is flagged as a temporary log file

Using ON-Monitor (UNIX)

The **Status > Logs** option displays much of the same information for logical-log files as the **onstat -l** option displays. In addition, a column contains the dbspace in which each logical-log file is located.

Monitoring Log-Backup Status

To monitor the status of the logs and to see which logs have been backed up, use the **onstat -l** command. A status flag of `B` indicates that the log has been backed up.

Allocating Log Files

When you initialize or restart the database server, it creates the number of logical-log files that you specify in the LOGFILES configuration parameter. These log files are the size that you specify in the LOGSIZE parameter.

Adding Logs Dynamically

The DYNAMIC_LOGS configuration parameter determines when the database server dynamically adds a logical-log file. When you use the default value of `2` for DYNAMIC_LOGS, the database server dynamically adds a new log file and sets off an alarm if the next active log file contains the beginning of the oldest open transaction.

The database server checks the logical-log space at these points:

- After switching to a new log file
- At the beginning of the transaction-cleanup phase of logical recovery

If the DYNAMIC_LOGS parameter is set to `1` and the next active log file contains records from an open transaction, the database server prompts you to add a log file manually and sets off an alarm. After you add the log file, the database server resumes processing the transaction.

If the DYNAMIC_LOGS parameter is set to `0` and the logical log runs out of space during a long transaction rollback, the database server can hang. (The long transaction prevents the first logical-log file from becoming free and available for reuse.) To fix the problem, set DYNAMIC_LOGS to `2` and restart the database server. Then the long transaction should be able to complete.

For more information, see “Monitoring Events for Dynamically Added Logs” on page 14-15 and “Setting High-Watermarks for Rolling Back Long Transactions” on page 14-16.

Size and Number of Dynamically Added Log Files

The purpose of dynamic logs is to add enough log space to allow transactions to roll back. When dynamically adding a log file, the database server uses the following factors to calculate the size of the log file:

- Average log size
- Amount of contiguous space available

If the logical log is low on space, the database server adds as many log files as needed to allow the transaction to roll back. The number of log files is limited by:

- The maximum number of log files supported
- The amount of disk space for the log files
- The amount of free contiguous space in the root chunk

If the database server stops adding new log files because it is out of disk space, it writes an error message and sets off an alarm. Add a dbspace or chunk to an existing dbspace. Then the database server automatically resumes processing the transaction.

The reserve pages in the root chunk store information about each log file. The extents that contain this information expand as more log files are added. The root chunk requires two extents of 1.4 megabytes each to track 32,767 log files, the maximum number supported.

If during reversion, the chunk reserve page extent is allocated from a non-root chunk, the server attempts to put it back in the root chunk. If not enough space is available in the root chunk, the reversion fails. A message containing the required space displays in the online log. The required space needs to be freed from the root chunk before trying the reversion again.

Location of Dynamically Added Log Files

The database server allocates log files in dbspaces, in the following search order. A dbspace becomes critical if it contains logical-log files or the physical log.

Pass	Allocate Log File In
1	The dbspace that contains the newest log files (If this dbspace is full, the database server searches other dbspaces.)
2	Mirrored dbspace that contains log files (but excluding the root dbspace)
3	All dbspaces that already contain log files (excluding the root dbspace)
4	The dbspace that contains the physical log
5	The root dbspace
6	Any mirrored dbspace
7	Any dbspace

If you do not want to use this search order to allocate the new log file, you must set the DYNAMIC_LOGS parameter to 1 and execute **onparams -a -i** with the location you want to use for the new log. For details, refer to "Monitoring Events for Dynamically Added Logs" on page 14-15.

Adding Logical-Log Files Manually

You might add logical-log files manually for the following reasons:

- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- To enable an open transaction to roll back
- As part of moving logical-log files to a different dbspace

Warning: You cannot do the following:

- Add a log file to a blobspace or sbspace.
- Add logical or physical logs to dbspaces that have non-default page sizes.

Add logical-log files one at a time, up to a maximum of 32,767 files, to any dbspace. As soon as you add a log file to a dbspace, it becomes a critical dbspace. You can add a logical-log file during a storage space backup.

The two ways you can add a logical-log file are:

- To the end of the file list using the **onparams -a** command or ISA
- After the current logical-log file using the **onparams -a -i** command or ISA

To add a logical-log file using onparams

1. Login as user **informix** or **root** on UNIX or as a member of the **Informix-Admin** group on Windows.
2. Ensure that the database server is in online, administration, or quiescent, or cleanup phase of fast-recovery mode.

   The database server writes the following message to the log during the cleanup phase:

   ```
   Logical recovery has reached the transaction cleanup phase.
   ```
3. Decide whether you want to add the log file to the end of the log file list or after the current log file.

   You can insert a log file after the current log file regardless of the DYNAMIC_LOGS parameter value. Adding a log file of a new size does not change the value of LOGSIZE.

   a. The following command adds a logical-log file to the end of the log file list in the **logspace** dbspace, using the log-file size specified by the LOGSIZE configuration parameter:

      ```
      onparams -a -d logspace
      ```

   b. The following command inserts a 1000-kilobyte logical-log file after the current log file in the **logspace** dbspace:

      ```
      onparams -a -d logspace -s 1000 -i
      ```

   c. To add a logical-log file with a new size (in this case, 250 kilobytes), execute the following command:

      ```
      onparams -a -d logspace -s 250
      ```
4. Use **onstat -l** to check the status of the log files. The status of the new log file is **A** and is immediately available.
5. The next time you need to back up data, perform a level-0 backup of the root dbspace and the dbspaces that contain the new log files.

Although you no longer need to back up immediately after adding a log file, your next backup should be level-0 because the data structures have changed. For information on backing up data, refer to the *IBM Informix Backup and Restore Guide*.

For more information on using **onparams** to add a logical-log file, see the *IBM Informix Administrator's Reference*.

To add a logical-log file using ISA

1. Select **Logs > Logical** and click **Add Log File**.
2. Use **onstat -l** to check the status of the log files.

   For more information, see the ISA online help.

To add a logical-log file with ON-Monitor (UNIX)

1. Follow the instructions on adding a log file in "Adding Logical-Log Files Manually" on page 14-11, except use ON–Monitor instead of **onparams**.
2. Select **Parameters > Add-Log** to add a logical-log file.
3. In the field labelled **Dbspace Name**, enter the name of the dbspace where the new logical-log file will reside.

   The size of the log file automatically appears in the **Logical Log Size** field. The new log file is always the value specified by LOGSIZE.

Dropping Logical-Log Files

To drop a logical-log file and increase the amount of the disk space available within a dbspace, you can use **onparams** or ISA. The database server requires a minimum of three logical-log files at all times. You cannot drop a log if your logical log is composed of only three log files.

The rules for dropping log files have changed:

- If you drop a log file that has never been written to (status **A**), the database server deletes it and frees the space immediately.
- If you drop a used log file (status **U-B**), the database server marks it as deleted (**D**). After you take a level-0 backup of the dbspaces that contain the log files and the root dbspace, the database server deletes the log file and frees the space.
- You cannot drop a log file that is currently in use or contains the last checkpoint record (status **C** or **L**).

To drop a logical-log file with onparams

1. Ensure that the database server is in online, administration, or quiescent mode.
2. Execute the following command to drop a logical-log file whose log file number is 21:

   ```
   onparams -d -l 21
   ```

   Drop log files one at a time. You must know the log file number of each logical log that you intend to drop.
3. If the log file has a status of newly Added (**A**), it is dropped immediately.

   If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.

   This backup prevents the database server from using the dropped log files during a restore and ensures that the reserved pages contain information about the current number of log files.

For information on using **onparams** to drop a logical-log file, see the *IBM Informix Administrator's Reference*.

For information on using **onlog** to display the logical-log files and unique ID numbers, see "Displaying Logical-Log Records" on page 14-15.

To drop a logical-log file with ON_Monitor (UNIX)

1. Ensure that the database server is in online, administration, or quiescent mode.
2. To drop a logical-log file, select **Parameters > Drop-Log**.
3. If the log file has a status of newly Added (**A**), it is dropped immediately.
   If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.

Tip: If the root dbspace has never been backed up, you can drop a **used** log file immediately.

Changing the Size of Logical-Log Files

If you want to change the size of the log files, it is easier to add new log files of the desired size and then drop the old ones. You can change the size of logical-log files in the following ways:

- Use **onparams** with the **-s** option to add a new log file of a different size.
  See "Adding Logical-Log Files Manually" on page 14-11.
- Increase the LOGSIZE value in the ONCONFIG file if you want the database server to create larger log files.
  See "Changing Logging Configuration Parameters" on page 14-14.

Moving a Logical-Log File to Another Dbspace

You might want to move a logical-log file for performance reasons or to make more space in the dbspace, as explained in "Location of Logical-Log Files" on page 13-1. To find the location of logical-log files, see "Monitoring Logging Activity" on page 14-7. Although moving the logical-log files is not difficult, it can be time-consuming.

Moving logical-log files is a combination of two simpler actions:

- Dropping logical-log files from their current dbspace
- Adding the logical-log files to their new dbspace

The following procedure provides an example of how to move six logical-log files from the **root** dbspace to another dbspace, **dbspace_1**.

Warning: You cannot move logical and physical log files in dbspaces that have non-default page sizes.

To move the logical-log files out of the root dbspace (example)

1. Ensure that the database server is in online, administration, quiescent, or fast-recovery mode.
2. Add six new logical-log files to **dbspace_1**.
   See "Adding Logical-Log Files Manually" on page 14-11.

3. Take a level-0 backup of all storage spaces to free all log files except the current log file.

   (If you use **onbar -l -b -c**, you back up all log files including the current log file.) See "Freeing a Logical-Log File" on page 14-5.
4. Use **onmode -l** to switch to a new current log file.

   See "Switching to the Next Logical-Log File" on page 14-5.
5. Drop all six logical-log files in the root dbspace.

   You cannot drop the current logical-log file.

   See "Dropping Logical-Log Files" on page 14-12.
6. Create a level-0 backup of the root dbspace and **dbspace_1**.

   For more information, see the *IBM Informix Backup and Restore Guide*.

Changing Logging Configuration Parameters

You can use a text editor or ISA to change ONCONFIG parameters. This table shows the configuration parameters for logical logs. For more information, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

Configuration Parameter	Minimum Value	Default Value	Maximum Value
DYNAMIC_LOGS	0 or 1	2	2
LOGBUFF	2 * page size	64 KB	LOGSIZE value
LOGFILES	3 files	5 files	32,767 files
LOGSIZE	10000 KB on UNIX 500 KB on Windows	10000 KB	See the *IBM Informix Administrator's Reference*
LTXEHWM	LTXHWM value	90%	100%
LTXHWM	1%	80%	100%

Important: The change to LOGFILES does not take effect until you reinitialize or restart the disk space.

To change the logical-log configuration parameters in the ONCONFIG file

1. Bring the database server offline or into quiescent or administration mode.
2. Use ISA or a text editor to update the configuration parameters.

   The DYNAMIC_LOGS, LTXHWM, and LTXEHWM parameters are not in the **onconfig.std** file. To change the values of these parameters, add them to your ONCONFIG file.
3. Shut down and restart the database server.
4. Perform this step only if you are changing LOGFILES and want all of the log files to be in continuous space. (Normally you add and drop LOGFILES using the **onparams** utility.)
   a. Unload all the database server data. Do this because you cannot rely on storage space backups to unload and restore the data because a restore returns the parameters to their previous value.
   b. Reinitialize the database server disk space. See "Initializing Disk Space" on page 4-1.
   c. Re-create all databases and tables.
   d. Reload all the database server data.

For information on loading and unloading data, see the *IBM Informix Migration Guide*.

5. Back up the root dbspace to enable your changed logical logs.

Using ON-Monitor to Change LOGFILES (UNIX)

You can use ON–Monitor to change the values of LOGFILES.

To change these values

1. Unload all the database server data.

   You cannot rely on storage space backups to unload and restore the data because a restore returns the parameters to their previous value.
2. Select **Parameters > Initialize** to reinitialize disk space or modify disk-space parameters.
3. Change the value of LOGSIZE in the field labelled **Log.Log Size**, or change the value of LOGFILES in the field labelled **Number of Logical Logs**.
4. Proceed with the database server disk-space initialization.
5. Re-create all databases and tables.
6. Reload all the database server data.

   For information on loading and unloading data, see the *IBM Informix Migration Guide*.

Displaying Logical-Log Records

Use the **onlog** utility to display and interpret logical-log records. For information on using **onlog**, see the *IBM Informix Administrator's Reference*.

Monitoring Events for Dynamically Added Logs

Monitor the following event alarms that dynamically added log files trigger (see Table 14-1). When each alarm is triggered, a message is written to the message log. For more information, see the chapters on event alarms and configuration parameters in the *IBM Informix Administrator's Reference*.

You can include the **onparams** command to add log files in your alarm script for event class ID 27, log file required. Your script can also execute the **onstat -d** command to check for adequate space and execute the **onparams a -i** command with the location that has enough space. You must use the **-i** option to add the new log right after the current log file.

Table 14-1. Event Alarms for Dynamically Added Log Files

Class ID	Severity	Class Message	Message
26	3	Dynamically added log file *log_number*	This message displays when the database server dynamically adds a log file. Dynamically added log file *log_number* to DBspace *dbspace_number*.

Table 14-1. Event Alarms for Dynamically Added Log Files (continued)

Class ID	Severity	Class Message	Message
27	4	Log file required	This message displays when DYNAMIC_LOGS is set to 1 and the database server is waiting for you to add a log file. ALERT: The oldest logical log *log_number* contains records from an open transaction *transaction_address*. Logical logging will remain blocked until a log file is added. Add the log file with the **onparams -a** command, using the **-i** (insert) option, as in: `onparams -a -d dbspace -s size-i` Then complete the transaction as soon as possible.
28	4	No space for log file	ALERT: Because the oldest logical log *log_number* contains records from an open transaction *transaction_address*, the server is attempting to dynamically add a log file. But there is no space available. Please add a dbspace or chunk. Then complete the transaction as soon as possible.

Table 14-2 shows the actions that the database server takes for each setting of the DYNAMIC_LOGS configuration parameter.

Table 14-2. DYNAMIC_LOGS Settings

DYNAMIC_ LOGS	Meaning	Event Alarm	Wait to Add Log	Dynamic Log Add
2 (default)	Allows automatic allocation of new log files to prevent open transactions from hanging the system.	Yes (26, 28)	No	Yes
1	Allows manual addition of new log files.	Yes (27)	Yes	No
0	Does not allocate log files but issues the following message about open transactions: WARNING: The oldest logical-log file *log_number* contains records from an open transaction *transaction_address*, but the dynamic log feature is turned off.	No	No	No

Setting High-Watermarks for Rolling Back Long Transactions

The database server uses the LTXHWM and LTXEHWM configuration parameters to set high-watermarks for long transactions. If DYNAMIC_LOGS is set to 1 or 2, the default LTXHWM value is 80 percent and LTXEHWM is 90 percent. If DYNAMIC_LOGS is set to 0, the default LTXHWM value is 50 percent and the default LTXHEWM value is 60 percent.

If you decrease your high-watermark values, you increase the likelihood of long transactions. To compensate, allocate additional log space. For information on LTXHWM and LTXEHWM, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

Long-Transaction High-Watermark (LTXHWM)

The *long-transaction high-watermark* is the percentage of total log space that a transaction is allowed to span before it is rolled back. If the database server finds

an open transaction in the oldest used log file, it dynamically adds log files. Because the log space is increasing, the high-watermark expands outward. When the log space reaches the high-watermark, the database server rolls back the transaction. The transaction rollback and other processes also generate logical-log records. The database server continues adding log files until the rollback is complete to prevent the logical log from running out of space. More than one transaction can be rolled back if more than one long transaction exists.

For example, the database server has ten logical logs and LTXHWM is set to 98. A transaction begins in log file 1 and update activity fills logs 1 through 9. The database server dynamically adds log file 11 after log file 10. As long as the transaction does not complete, this process continues until the database server has added 40 log files. When the database server adds the fiftieth log, the transaction has caught up to the high-watermark and the database server rolls it back.

Exclusive Access, Long-Transaction High-Watermark (LTXEHWM)

The *exclusive-access, long-transaction high-watermark* occurs when the long transaction currently being rolled back is given *exclusive* access to the logical log. The database server dramatically reduces log-record generation. Only threads that are currently rolling back transactions and threads that are currently writing COMMIT records are allowed access to the logical log. Restricting access to the logical log preserves as much space as possible for rollback records that are being written by the user threads that are rolling back transactions.

Warning: If you set both *LTXHWM* and *LTXEHWM* to *100*, long transactions are never stopped. Therefore, you should set *LTXHWM* to below *100* for normal database server operations. Set *LTXHWM* to *100* to run scheduled transactions of unknown length. Set *LTXEHWM* to 100 if you never want to block other users while a long transaction is rolling back and you have ample disk space.

Adjusting the Size of Log Files to Prevent Long Transactions

Use larger log files when many users are writing to the logs at the same time. If you use small logs and long transactions are likely to occur, reduce the high-watermark. Set the LTXHWM value to `50` and the LTXEHWM value to `60`.

If the log files are too small, the database server might run out of log space while rolling back a long transaction. In this case, the database server cannot block fast enough to add a new log file before the last one fills. If the last log file fills, the system will hang and display an error message. To fix the problem, shut down and restart the database server. For details, see "Recovering from a Long Transaction Hang."

Recovering from a Long Transaction Hang

If your system has ample disk space and you want to perform transactions of unknown length, consider setting LTXHWM to `100` to force the database server to continue adding log files until you complete the transaction.

A transaction might hang because the database server has run out of disk space. The database server stops adding new log files, writes an error message, and sets off an alarm.

To continue the transaction

1. To continue the transaction, add a dbspace or chunk to a dbspace.
2. Resume processing the transaction.

If you cannot add more disk space to the database server, end the transaction.

To end the transaction

- Issue the **onmode -z** command.
- Shut down and restart the database server.

When the database server comes up in fast-recovery mode, the transaction is rolled back. Then perform the following steps:

To recover from a long transaction hang

1. Add more disk space or another disk until the transaction is successfully rolled back.
2. Perform a point-in time restore to a time before the long transaction began or early enough for the database server to roll back the transaction.
3. Drop the extra log files, dbspaces, or chunks from the database server instance.
4. Perform a complete level-0 backup to free the logical-log space.

Chapter 15. Physical Logging, Checkpoints, and Fast Recovery

In This Chapter

This chapter covers the three procedures that the database server uses to achieve data consistency:

- Physical logging
- Checkpoints
- Fast recovery

The *physical log* is a set of disk pages where the database server stores an unmodified copy of the page called a *before-image*. *Physical logging* is the process of storing a before-image of a page that the database server is going to change. A *checkpoint* refers to a point when the database server synchronizes the pages on disk with the pages in the shared-memory buffers. *Fast recovery* is an automatic procedure that restores the database server to a consistent state after it goes offline under uncontrolled conditions.

These procedures ensure that multiple, logically related writes are recorded as a unit, and that data in shared memory is periodically made consistent with data on disk.

For the tasks to manage and monitor the physical log and checkpoints, see Chapter 16, "Managing the Physical Log," on page 16-1.

Critical Sections

A *critical section* is a section of code (or machine instructions) that must be performed as a single unit. A critical section ensures the integrity of a thread by allowing it to execute a series of instructions before it is swapped out.

Physical Logging

Physical logging is the process of storing the pages that the database server is going to change before the changed pages are actually recorded on disk. Before the database server modifies certain pages in the shared-memory buffer pool, it stores before-images of the pages in the physical-log buffer in shared memory.

The database server maintains the before-image page in the physical-log buffer in shared memory for those pages until one or more page cleaners flush the pages to disk. The unmodified pages are available in case the database server fails or the backup procedure needs them to provide an accurate snapshot of the database server data. Fast recovery and database server backups use these snapshots.

The database server recycles the physical log at each checkpoint, except in the special circumstances. For more information on checkpoints, see "Checkpoints" on page 15-4.

Fast Recovery Use of Physically-Logged Pages

After a failure, the database server uses the before-images of pages to restore these pages on the disk to their state at the last checkpoint. Then the database server uses the logical-log records to return all data to physical and logical consistency, up to the point of the most-recently completed transaction. "Fast Recovery" on page 15-7 explains this procedure in more detail.

Backup Use of Physically-Logged Pages

When you perform a backup, the database server performs a checkpoint and coordinates with the physical log to identify the correct version of pages that belong on the backup. In a level-0 backup, the database server backs up all disk pages. For more details, see the *IBM Informix Backup and Restore Guide*.

Database Server Activity That Is Physically Logged

If multiple modifications were made to a page between checkpoints, typically only the first before-image is logged in the physical log.

The physical log is a cyclical log in which the pages within the physical log are used once per checkpoint. If the RTO_SERVER_RESTART configuration parameter is set, additional physical logging occurs to improve fast recovery performance.

Physical Recovery Messages

When fast recovery begins, the database server logs the following message with the name of the chunk and offset:

```
Physical recovery started at page chunk:offset.
```

When the fast recovery completes, the database server logs the following message with the number of pages examined and restored:

```
Physical recovery complete: number pages examined, number pages restored.
```

Physical Logging and Simple Large Objects

The database server pages in the physical log can be any database server page, including simple large objects in table spaces (tblspaces). Even overhead pages (such as chunk free-list pages, blobspace free-map pages, and blobspace bit-map pages) are copied to the physical log before data on the page is modified and flushed to disk.

Blobspace blobpages are not logged in the physical log. For more information about blobspace logging, see "Logging Blobspaces and Simple Large Objects" on page 13-6.

Physical Logging and Smart Large Objects

The user-data portion of smart large objects is not physically logged. However, the metadata is physically logged. For information on smart large objects, see "Sbspaces" on page 9-12.

Size and Location of the Physical Log

This section describes how to configure the size and location of the physical log.

Specifying the Location of the Physical Log

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace. You have no initial control over this placement. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize disk contention), you can move the physical log out of the root dbspace to another dbspace, preferably to a disk that does not contain active tables or the logical-log files.

Recommendation: Locate critical dbspaces on fault-tolerant storage devices. If the storage that the physical log resides in is not fault-tolerant, use Dynamic Server mirroring for the dbspace that contain the physical log. This protects the database if the storage devices fails.

Strategy for Estimating the Size of the Physical Log

The size of the physical log depends on two factors:

- The rate at which transactions generate physical log activity
- Whether or not you set the RTO_SERVER_RESTART configuration parameter

The rate at which transactions generate physical log activity can impact checkpoint performance. During checkpoint processing, if the physical log starts getting too full as transactions continue to generate physical log data, the database server blocks transactions to allow the checkpoint to complete and to avoid a physical log overflow.

To avoid transaction blocking, the database server must have enough physical log space to contain all of the transaction activity that occurs during checkpoint processing. Checkpoints are triggered when the physical log is 75 percent full. Checkpoint processing must complete before the remaining 25 percent of the physical log is used. Transaction blocking occurs as soon as the system detects a potential for a physical log overflow, because every active transaction could generate physical log activity.

Dynamic Server might also trigger checkpoints if a large number of dirty partitions exist, even if the physical log is not 75 percent full, because flushing the modified partition data to disk requires physical log space.

For example, suppose you have a one gigabyte physical log and 1000 active transactions. 1000 active transactions have the potential to generate approximately 80 megabytes of physical log activity if every transaction is in a critical section simultaneously. When 750 megabytes of the physical log fills, the database server triggers a checkpoint. If the checkpoint has not completed by the time the 920 megabytes of the physical log are used, transaction blocking occurs until the checkpoint completes. If transaction blocking takes place, the server automatically triggers more frequent checkpoints to avoid transaction blocking. You can disable the generation of automatic checkpoints.

For more information on checkpoint processing and automatic checkpoints, see "Checkpoints" on page 15-4.

The second factor to consider when estimating the size of the physical log depends on the your use of the RTO_SERVER_RESTART configuration parameter to specify a target amount of time for fast recovery. If you do not need to consider fast recovery time, you do not need to enable the RTO_SERVER_RESTART

configuration parameter. If you specify a value for the RTO_SERVER_RESTART configuration parameter, transaction activity generates additional physical log activity.

Typically, this additional physical log activity has little or no performance impact on transaction performance. The extra logging is used to assist the bufferpool during fast recovery, so that log replay performs optimally. If the physical log is considerably larger than the combined sizes of all buffer pools, page flushing and page faulting occur during fast recovery. The page flushing and page faulting substantially reduce fast recovery performance, and the database server cannot maintain the RTO_SERVER_RESTART policy.

For systems with less that four gigabytes of buffer pool space, the physical log can be sized at 110 percent of the combined size of all the buffer pools. For larger buffer pools, start with four gigabytes of physical log space and then monitor checkpoint activity. If checkpoints occur too frequently and appear to impact performance, increase the physical log size.

A rare condition, called a physical-log overflow, can occur when the database server is configured with a small physical log and has a large number of users. Following the size guidelines above, helps avoid physical-log overflow. The database server generates performance warnings to the message log whenever it detects suboptimal configurations.

You can use the **onstat -g ckp** command to display configuration recommendations if a suboptimal configuration is detected.

Physical-Log Overflow When Transaction Logging Is Turned Off

The physical log can overflow if you use simple large objects or smart large objects in a database with transaction logging turned off, as the following example shows.

When the database server processes simple large objects, each portion of the simple large object that the database server stores on disk can be logged separately, allowing the thread to exit the critical sections of code between each portion. However, if logging is turned off, the database server must carry out all operations on the simple large object in one critical section. If the simple large object is large and the physical log small, this scenario can cause the physical logs to become full. If this situation occurs, the database server sends the following message to the message log:

```
Physical log file overflow
```

The database server then initiates a shutdown. For the suggested corrective action, refer to your message log.

Checkpoints

Periodically, the database server flushes transactions and data within the buffer pool to disk. Until the transactions and data are flushed to disk, the data and transactions are in a state of flux. Instead of forcing every transaction to disk immediately after a transaction is completed, the database server writes transactions to the logical log. The database server logs the transactions as they occur. In the event of a system failure, the server:

- Replays the log to redo and restore the transactions.

- Returns the database to a state consistent with the state of the database system at the time of the failure.

To facilitate the restoration or logical recovery of a database system, the database server generates a consistency point, called a *checkpoint*. A checkpoint is a point in time in the log when a known and consistent state for the database system is established. Typically, a checkpoint involves recording a certain amount of information so that, if a failure occurs, the database server can restart at that established point.

The purpose of a checkpoint is to periodically move the restart point forward in the logical log. If checkpoints did not exist and a failure occurred, the database server would need to process all the transactions that were recorded in the logical log since the system restarted.

A checkpoint can occur in one of these situations:

- When specific events occur. For example, a checkpoint occurs whenever a dbspace is added to the server or a database backup is performed.

  Typically, these types of events trigger checkpoints that block transaction processing. Therefore, these checkpoints are called *blocking* checkpoints.
- When resource limitations occur. For example, a checkpoint is required for each span of the logical log space to guarantee that the log has a checkpoint at which to begin fast recovery. The database server triggers a checkpoint when the physical log is 75 percent full to avoid physical log overflow.

  Checkpoints triggered by resource limitations usually do not block transactions. Therefore, these checkpoints are called *nonblocking checkpoints*.

  However, if the database server begins to run out of resources during checkpoint processing, transaction blocking occurs in the midst of checkpoint processing to make sure that the checkpoint completes before a resource is depleted. If transactions are blocked, the server attempts to trigger checkpoints more frequently to avoid transaction blocking during checkpoint processing. For more information, see "Strategy for Estimating the Size of the Physical Log" on page 15-3.

Automatic checkpoints cause the database server to trigger more frequent checkpoints to avoid transaction blocking. Automatic checkpoints attempt to monitor system activity and resource usage (physical and logical log usage along with how dirty the buffer pools are) to trigger checkpoints in a timely manner so that the processing of the checkpoint can complete before the physical or logical log is depleted.

The database server generates at least one automatic checkpoint for each span of the logical-log space. This guarantees the existence of a checkpoint where fast recovery can begin.

Use the AUTO_CKPTS configuration parameter to enable or disable automatic checkpoints when the database server starts. (You can dynamically enable or disable automatic checkpoints by using **onmode -wm** or **onmode -wf**.)

Manual checkpoints are event-based checkpoints that you can initiate.

The database server provides two methods for determining how long fast recovery will take in the event of an unplanned outage.

- Use the CKPTINTVL configuration parameter to specify how frequently the server triggers checkpoints.
- Use the RTO_SERVER_RESTART configuration parameter to specify how long fast recovery should take.

  When you use the RTO_SERVER_RESTART configuration parameter:

  – The database server ignores the CKPTINTVL configuration parameter.
  – The database server monitors the physical and logical log usage to estimate the duration of fast recovery. If the server estimates that fast recovery will exceed the time specified in the RTO_SERVER_RESTART configuration parameter, the server automatically triggers a checkpoint.

The RTO_SERVER_RESTART configuration parameter is intended to be a target amount of time, not a guaranteed amount of time.

Several factors that can increase restart time can also influence fast recovery time. These factors include rolling back long transactions that were active at the time of an unplanned outage.

For more information about the RTO_SERVER_RESTART and AUTO_CKPTS configuration parameters, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

LRU Values for Flushing a Buffer Pool Between Checkpoints

The LRU values for flushing a buffer pool between checkpoints are not critical for checkpoint performance. The **lru_max_dirty** and **lru_min_dirty** values, which are set in the BUFFERPOOL configuration parameter, are usually necessary only for maintaining enough clean pages for page replacement. Start by setting **lru_min_dirty** to `70` and **lru_max_dirty** to `80`.

If transactions are blocked during a checkpoint, the database server subsequently attempts to increase checkpoint frequency to eliminate the transaction being blocked. When the server searches for a free page to perform page replacement and a foreground write occurs, the server subsequently automatically increases the LRU flushing frequency to prevent this event from occurring again. When the database server completes page replacement and finds a frequently accessed page, the server automatically increases LRU flushing. Any automatic adjustments to LRU flushing do not persist to the ONCONFIG file.

For more information on monitoring and tuning checkpoint parameters and information on LRU tuning and adjustments, see the *IBM Informix Dynamic Server Performance Guide*.

Checkpoints During Backup

If you perform a backup, the database server runs a checkpoint and flushes all changed pages to the disk. If you perform a restore, the database server reapplies all logical-log records.

For information on ON–Bar or **ontape**, see the *IBM Informix Backup and Restore Guide*.

Fast Recovery

Fast recovery is an automatic, fault-tolerant feature that the database server executes every time that it moves from offline to quiescent, administration, or online mode. You do not need to take any administrative actions for fast recovery; it is an automatic feature.

The fast-recovery process checks if, the last time that the database server went offline, it did so in uncontrolled conditions. If so, fast recovery returns the database server to a state of physical and logical consistency.

If the fast-recovery process finds that the database server came offline in a controlled manner, the fast-recovery process terminates, and the database server moves to online mode.

See "Fast Recovery After a Checkpoint" on page 15-8.

Need for Fast Recovery

Fast recovery restores the database server to physical and logical consistency after any failure that results in the loss of the contents of memory for the database server. For example, the operating system fails without warning. System failures do not damage the database but instead affect transactions that are in progress at the time of the failure.

Situations When Fast Recovery Is Initiated

Every time that the administrator brings the database server to quiescent, administration, or online mode from offline mode, the database server checks to see if fast recovery is needed.

As part of shared-memory initialization, the database server checks the contents of the physical log. The physical log is empty when the database server shuts down under control. The move from online mode to quiescent mode includes a checkpoint, which flushes the physical log. Therefore, if the database server finds pages in the physical log, the database server clearly went offline under uncontrolled conditions, and fast recovery begins.

Fast Recovery and Buffered Logging

If a database uses buffered logging (as described in "Buffered Transaction Logging" on page 11-6), some logical-log records associated with committed transactions might not be written to the logical log at the time of the failure. If this occurs, fast recovery cannot restore those transactions. Fast recovery can restore only transactions with an associated COMMIT record stored in the logical log on disk. (For this reason, buffered logging represents a trade-off between performance and data vulnerability.)

Possible Physical Log Overflow During Fast Recovery

During fast recovery, the physical log can overflow. If this occurs, the database server tries to extend the physical log space to a disk file named **plog_extend.***servernum*. The default location of this file is **$INFORMIXDIR/tmp**.

Use the ONCONFIG parameter **PLOG_OVERFLOW_PATH** to define the location for creating this file.

The database server removes the **plog_extend.*servernum*** file when the first checkpoint is performed during a fast recovery.

Fast Recovery and No Logging

For databases or tables that do not use logging, fast recovery restores the database to its state at the time of the most recent checkpoint. All changes made to the database since the last checkpoint are lost.

Fast Recovery After a Checkpoint

Fast recovery returns the database server to a consistent state as part of shared-memory initialization. All committed transactions are restored, and all uncommitted transactions are rolled back.

Fast recovery occurs in the following steps:

1. The database server uses the data in the physical log to return all disk pages to their condition at the time of the most recent checkpoint. This point is known as *physical consistency*.
2. The database server locates the most recent checkpoint record in the logical-log files.
3. The database server rolls forward all logical-log records written after the most recent checkpoint record.
4. The database server rolls back all uncommitted transactions. Some XA transactions might be unresolved until the XA resource manager is available.

The following sections describe each step in detail:

- "The Server Returns to the Last-Checkpoint State"
- "The Server Locates the Checkpoint Record in the Logical Log" on page 15-9
- "The Server Rolls Forward Logical-Log Records" on page 15-9
- "The Server Rolls Back Uncommitted Transactions" on page 15-9

The Server Returns to the Last-Checkpoint State

To return all disk pages to their condition at the time of the most recent checkpoint, the database server writes the before-images stored in the physical log to shared memory and then back to disk. Each before-image in the physical log contains the address of a page that was updated after the checkpoint. When the database server writes each before-image page in the physical log to shared memory and then back to disk, changes to the database server data since the time of the most recent checkpoint are undone. The database server is now physically consistent. Figure 15-1 illustrates this step.

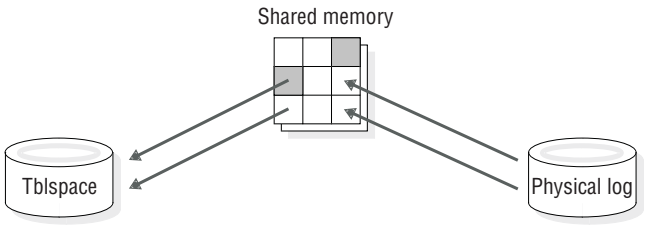


Figure 15-1. Writing All Remaining Before-Images in the Physical Log Back to Disk

The Server Locates the Checkpoint Record in the Logical Log

After returning to the last checkpoint state, the database server locates the address of the most recent checkpoint record in the logical log. The most recent checkpoint record is guaranteed to be in the logical log on disk.

The Server Rolls Forward Logical-Log Records

After locating the checkpoint record in the logical log, the database server rolls forward the logical-log records that were written after the most recent checkpoint record. This action reproduces all changes to the databases since the time of the last checkpoint, up to the point at which the uncontrolled shutdown occurred. Figure 15-2 illustrates this step.

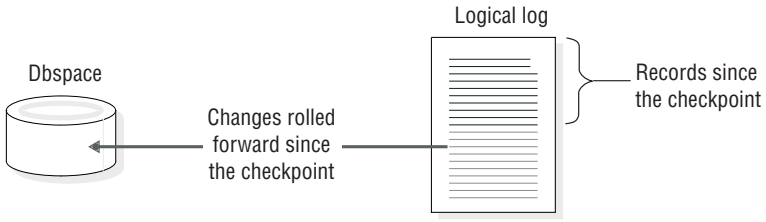


Figure 15-2. Rolling Forward the Logical-Log Records Written Since the Most Recent Checkpoint

The Server Rolls Back Uncommitted Transactions

After rolling the logical-log records forward, the database server rolls back all logical-log records for transactions that were not committed at the time the system failed. All databases are logically consistent because all committed transactions are rolled forward and all uncommitted transactions are rolled back. Some XA transactions might be unresolved until the XA resource manager is available.

Transactions that have completed the first phase of a two-phase commit are exceptional cases. For more information, see "How the Two-Phase Commit Protocol Handles Failures" on page 25-6.

Because one or more transactions possibly spanned several checkpoints without being committed, this rollback procedure might read backward through the logical log, past the most recent checkpoint record. All logical-log files that contain records for open transactions are available to the database server because a log file is not freed until all transactions that it contains are closed.

Figure 15-3 on page 15-10 illustrates the rollback procedure. Here, uncommitted changes are rolled back from the logical log to a dbspace on a particular disk. When fast recovery is complete, the database server returns to quiescent, administration, or online mode.

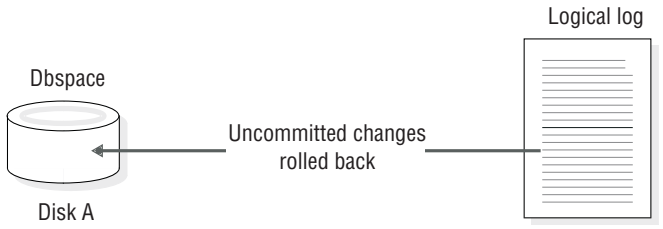


Figure 15-3. Rolling Back All Incomplete Transactions

Chapter 16. Managing the Physical Log

In This Chapter

This chapter describes the following procedures:

- Changing the location and size of the physical log
- Monitoring the physical log, physical-log buffers, and logical-log buffers
- Monitoring and forcing checkpoints

See Chapter 15, "Physical Logging, Checkpoints, and Fast Recovery," on page 15-1 for background information.

Changing the Physical-Log Location and Size

You can change your physical-log location or size in either of these ways:

- Using the **onparams** utility from the command line
- Using ON–Monitor

Log in as user **informix** or root on UNIX or as a member of the **Informix-Admin** group on Windows when you make the changes.

You can move the physical-log file to try to improve performance. When the database server initializes disk space, it places the disk pages allocated for the logical log and the physical log in the root dbspace. You might improve performance by moving the physical log, the logical-log files, or both to other dbspaces.

Note: You cannot add logical or physical logs to dbspaces that have non-default page sizes.

For advice on where to place the physical log, see "Specifying the Location of the Physical Log" on page 15-3. For advice on sizing the physical log, see "Size and Location of the Physical Log" on page 15-2. To obtain information about the physical log, see "Monitoring Physical and Logical-Logging Activity" on page 16-2.

Checking For Adequate Contiguous Space

The space allocated for the physical log must be contiguous. If insufficient resources for the physical log exist when you use **oninit -i** to initialize the database server, the initialization fails.

When changing the physical log size or location, if the target dbspace does not contain adequate contiguous space, the server does not change the physical log.

You can check whether adequate contiguous space is available with the **oncheck -pe** option. For information on using the **oncheck -ce** and **-pe** options to check the chunk-free list, see the *IBM Informix Administrator's Reference*.

Using onparams to Change Physical-Log Location or Size

To change the size and location of the physical log, execute the following command after you bring the database server to quiescent or administration mode:

```
onparams -p -s size -d dbspace -y
```

size is the new size of the physical log in kilobytes

dbspace
specifies the dbspace where the physical log is to reside

The following example changes the size and location of the physical log. The new physical-log size is 400 kilobytes, and the log will reside in the **dbspace6** dbspace:

```
onparams -p -s 400 -d dbspace6 -y
```

For information on using the **onparams** utility to modify the physical log, see the *IBM Informix Administrator's Reference*.

Using ON-Monitor to Change Physical-Log Location or Size (UNIX)

To change the log size or dbspace location, or both, using ON-Monitor, select **Parameters > Physical-Log**.

Monitoring Physical and Logical-Logging Activity

Monitor the physical log to determine the percentage of the physical-log file that gets used before a checkpoint occurs. This information allows you to find the optimal size of the physical-log file. It should be large enough that the database server does not have to force checkpoints too frequently and small enough to conserve disk space and guarantee fast recovery.

Monitor physical-log and logical-log buffers to determine if they are the optimal size for the current level of processing. The important statistic to monitor is the pages-per-disk-write statistic. For more information on tuning the physical-log and logical-log buffers, see your *IBM Informix Performance Guide*.

To monitor the physical-log file, physical-log buffers, and logical-log buffers, use the following commands.

Utility	Command	Additional Information
Command line or ISA	**onstat -l**	The first line displays the following information for each physical-log buffer: • The number of buffer pages used (**bufused**) • The size of each physical log buffer in pages (**bufsize**) • The number of pages written to the buffer (**numpages**) • The number of writes from the buffer to disk (**numwrits**) • The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**) The second line displays the following information about the physical log: • The page number of the first page in the physical-log file (**phybegin**) • The size of the physical-log file in pages (**physize**) • The current position in the log where the next write occurs, specified as a page number (**physpos**) • The number of pages in the log that have been used (**phyused**) • The percentage of the total physical-log pages that have been used (**%used**) The third line displays the following information about each logical-log buffer: • The number of buffer pages used (**bufused**) • The size of each logical-log buffer in pages (**bufsize**) • The number of records written to the buffer (**numrecs**) • The number of pages written to the buffer (**numpages**) • The number of writes from the buffer to disk (**numwrits**) • The ratio of records to pages in the buffer (**recs/pages**) • The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)
Command line or ISA	**onparams -p**	Moves or resizes the physical log.
Command line or ISA	**onmode -l**	Advances to the next logical-log file.
ISA	**Logs > Logical**	Click **Advance Log File**.

Note: For more information on and an example of **onstat -l** output, see the *IBM Informix Dynamic Server Administrator's Reference*.

For information on using SQL administration API commands instead of some **onparams** and **onmode** commands, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1 and the *IBM Informix Guide to SQL: Syntax*.

Monitoring Checkpoint Information

Monitor checkpoint activity to determine basic checkpoint information. This information includes the number of times that threads had to wait for the checkpoint to complete. This information is useful for determining if the checkpoint interval is appropriate. For information on tuning the checkpoint interval, see your *IBM Informix Dynamic Server Performance Guide*.

To monitor checkpoints, use the following commands.

Utility	Command	Additional Information
Command line or ISA	**onstat -m**	View the last 20 lines in the message log. If a checkpoint message does not appear in the last 20 lines, read the message log directly with a text editor. The database server writes individual checkpoint messages to the log when the checkpoint ends. If a checkpoint occurs, but the database server has no pages to write to disk, the database server does not write any messages to the message log.
Command line or ISA	**onstat -p**	Obtains these checkpoint statistics: • **numckpts**: Number of checkpoints that occurred since the database server was brought online. • **ckptwaits**: Number of times that a user thread waits for a checkpoint to finish. The database server prevents a user thread from entering a critical section during a checkpoint.
ON–Monitor (UNIX)	**Status > Profile**	The **Checkpoints** and **Check Waits** fields display the same information as the **numckpts** and **ckpwaits** fields in **onstat -p**.

Turning Checkpoint Tuning On or Off

To turn automatic checkpoint tuning on, issue an **onmode –wf** AUTO_CKPTS=1 command. To turn automatic checkpoint tuning off, issue an **onmode –wf** AUTO_CKPTS=0 command.

Forcing a Checkpoint

To force a checkpoint, execute one of the following commands.

Utility	Command	Additional Information
Command line or ISA	**onmode -c**	None.
ISA	**Logs > Logical**	Click **Force Checkpoint**.
ON–Monitor (UNIX)	**Force-Ckpt** option from the main menu	The time in the **Last Checkpoint Done** field does not change until a checkpoint occurs. The **Last Checkpoint Check** field shows the time of the last checkpoint check. If no modifications have been made since the time of the last checkpoint, the database server does not perform a checkpoint.

Force a checkpoint in any of the following situations:

- To free a logical-log file that contains the most recent checkpoint record and that is backed up but not yet released (**onstat -l** status of U-B-L or U-B)

- Before you issue **onmode -sy** to place the database server in quiescent mode
- After building a large index, if the database server terminates before the next checkpoint (The index build will restart the next time that you restart the database server.)
- If a checkpoint has not occurred for a long time and you are about to attempt a system operation that might interrupt the database server
- If foreground writes are taking more resources than you want (force a checkpoint to bring this down to zero temporarily)
- Before you execute **dbexport** or unload a table, to ensure physical consistency of all data before you export or unload it
- After you perform a large load of tables using PUT or INSERT statements (Because table loads use the buffer cache, forcing a checkpoint cleans the buffer cache.)

For information on using SQL administration API commands instead of some **onmode** commands, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1 and the *IBM Informix Guide to SQL: Syntax*.

Using Server-Provided Checkpoint Statistics

The database server provides history information on the previous twenty checkpoints. You can access this information through the SMI **sysckpthist** and **sysckptinfo** tables.

Using SMI Tables

Query the **sysprofile** table to obtain statistics on the physical-log and logical-log buffers. The **sysprofile** table also provides the same checkpoint statistics that are available from the **onstat -p** option. These rows contain the following statistics.

Row **Description**

plgpagewrites
Number of pages written to the physical-log buffer

plgwrites
Number of writes from the physical-log buffer to the physical log file

llgrecs Number of records written to the logical-log buffer

llgpagewrites
Number of pages written to the logical-log buffer

llgwrites
Number of writes from the logical-log buffer to the logical-log files

numckpts
Number of checkpoints that have occurred since the database server was brought online)

ckptwaits
Number of times that threads waited for a checkpoint to finish to enter a *critical section* during a checkpoint

value Values for **numckpts** and **ckptwaits**

Turning Automatic LRU Tuning On or Off

If the RTO_SERVER_RESTART configuration parameter is set, the database server automatically triggers checkpoints so that it can bring the server online within the specified time. The database server prints warning messages in the message log if the server cannot meet the RTO_SERVER_RESTART policy.

You use the AUTO_LRU_TUNING configuration parameter to enable or disable automatic LRU tuning when the database server starts.

To turn off automatic LRU tuning for a particular session, issue an **onmode –wm** AUTO_LRU_TUNING=*0* command.

To turn on automatic LRU tuning after turning it off during a session, issue an **onmode –wm** AUTO_LRU_TUNING=*1* command

Automatic LRU tuning changes affect all buffer pools and adjust **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

For more information on LRU tuning, see the *IBM Informix Dynamic Server Performance Guide*.

Part 4. Fault Tolerance

Chapter 17. Mirroring

In This Chapter

This chapter describes the database server mirroring feature. For instructions on how to perform mirroring tasks, refer to Chapter 18, "Using Mirroring," on page 18-1.

Mirroring

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace, blobspace, or sbspace with an equal-sized *mirror chunk*.

Every write to the primary chunk is automatically accompanied by an identical write to the mirror chunk. This concept is illustrated in Figure 17-1. If a failure occurs on the primary chunk, mirroring enables you to read from and write to the mirror chunk until you can recover the primary chunk, all without interrupting user access to data.

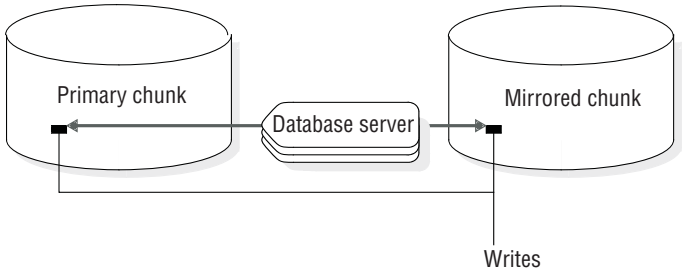


Figure 17-1. Writing Data to Both the Primary Chunk and the Mirror Chunk

Mirroring is not supported on disks that are managed over a network. The same database server instance must manage all the chunks of a mirrored set.

Benefits of Mirroring

If media failure occurs, mirroring provides the database server administrator with a means of recovering data without having to take the database server offline. This feature results in greater reliability and less system downtime. Furthermore, applications can continue to read from and write to a database whose primary chunks are on the affected media, provided that the chunks that mirror this data are located on separate media.

Any critical database should be located in a mirrored dbspace. Above all, the root dbspace, which contains the database server reserved pages, should be mirrored.

Costs of Mirroring

Disk-space costs as well as performance costs are associated with mirroring. The disk-space cost is due to the additional space required for storing the mirror data. The performance cost results from having to perform writes to both the primary and mirror chunks. The use of multiple virtual processors for disk writes reduces this performance cost. The use of *split reads*, whereby the database server reads data from either the primary chunk or the mirror chunk, depending on the location

of the data within the chunk, actually causes performance to improve for read-only data. For more information on how the database server performs reads and writes for mirror chunks, see "Actions During Processing" on page 17-4.

Consequences of Not Mirroring

If you do not mirror your dbspaces, the frequency with which you have to restore from a storage-space backup after media failure increases.

When a mirror chunk suffers media failure, the database server reads exclusively from the chunk that is still online until you bring the down chunk back online. On the other hand, when an *mirrored* chunk goes down, the database server cannot access the data stored on that chunk. If the chunk contains logical-log files, the physical log, or the root dbspace, the database server goes offline immediately. If the chunk does not contain logical-log files, the physical log, or the root dbspace, the database server can continue to operate, but threads cannot read from or write to the down chunk. If an unmirrored chunk goes down, you must restore it by recovering the dbspace from a backup.

Data to Mirror

Ideally, you should mirror all of your data. If disk space is an issue, however, you might not be able to do so. In this case, select certain critical chunks to mirror.

Critical chunks always include the chunks that are part of the root dbspace, the chunk that stores the logical-log files, and the chunk that stores the physical logs. If any one of these critical chunks fail, the database server goes offline immediately.

If some chunks hold data that is critical to your business, give these chunks high priority for mirroring.

Also give priority for mirroring to chunks that store frequently used data. This action ensures that the activities of many users would not be halted if one widely used chunk goes down.

Alternatives to Mirroring

Mirroring, as discussed in this manual, is a database server feature. Your operating system or hardware might provide alternative mirroring solutions.

If you are considering a mirroring feature provided by your operating system instead of database server mirroring, compare the implementation of both features before you decide which to use. The slowest step in the mirroring process is the actual writing of data to disk. The database server strategy of performing writes to mirror chunks in parallel helps to reduce the time required for this step. (See "Disk Writes to Mirror Chunks" on page 17-4.) In addition, database server mirroring uses split reads to improve read performance. (See "Disk Reads from Mirror Chunks" on page 17-5.) Operating-system mirroring features that do not use parallel mirror writes and split reads might provide inferior performance.

Nothing prevents you from running database server mirroring and operating-system mirroring at the same time. They run independently of each other. In some cases, you might decide to use both the database server mirroring and the mirroring feature provided by your operating system. For example, you might have both database server data and other data on a single disk drive. You could use the operating-system mirroring to mirror the other data and database server mirroring to mirror the database server data.

Logical Volume Managers

Logical volume managers are an alternative mirroring solution. Some operating-system vendors provide this type of utility to have multiple disks appear as one file system. Saving data to more than two disks gives you added protection from media failure, but the additional writes have a performance cost.

Hardware Mirroring

Another solution is to use hardware mirroring such as RAID (redundant array of inexpensive disks). An advantage of this type of hardware mirroring is that it requires less disk space than database server mirroring does to store the same amount of data to prevent media failure.

Some hardware mirroring systems support *hot swapping*. You can swap a bad disk while keeping the database server online. Reducing I/O activity before performing a hot swap is recommended.

Important: If problems occur with the database server while using hardware mirroring, refer to the operating-system or disk documentation or technical support for assistance.

External Backup and Restore

If you use hardware disk mirroring, you can get your system online faster with external backup and restore than with conventional ON–Bar commands. For more information on external backup and restore, see the *IBM Informix Backup and Restore Guide*.

Mirroring Process

This section describes the mirroring process in greater detail. For instructions on how to perform mirroring operations such as creating mirror chunks, starting mirroring, changing the status of mirror chunks, and so forth, see Chapter 18, "Using Mirroring," on page 18-1.

Creation of a Mirror Chunk

When you specify a mirror chunk, the database server copies all the data from the primary chunk to the mirror chunk. This copy process is known as *recovery*. Mirroring begins as soon as recovery is complete.

The recovery procedure that marks the beginning of mirroring is delayed if you start to mirror chunks within a dbspace that contains a logical-log file. Mirroring for dbspaces that contain a logical-log file does not begin until you create a level-0 backup of the root dbspace. The delay ensures that the database server can use the mirrored logical-log files if the primary chunk that contains these logical-log files becomes unavailable during a dbspace restore.

The level-0 backup copies the updated database server configuration information, including information about the new mirror chunk, from the root dbspace reserved pages to the backup. If you perform a data restore, the updated configuration information at the beginning of the backup directs the database server to look for the mirrored copies of the logical-log files if the primary chunk becomes unavailable. If this new storage-space backup information does not exist, the database server is unable to take advantage of the mirrored log files.

For similar reasons, you cannot mirror a dbspace that contains a logical-log file while a dbspace backup is being created. The new information that must appear in the first block of the dbspace backup tape cannot be copied there after the backup has begun.

For more information on creating mirror chunks, refer to Chapter 18, "Using Mirroring," on page 18-1.

Mirror Status Flags

Dbspaces, blobspaces, and sbspaces have status flags that indicate whether they are mirrored or unmirrored.

You must perform a level-0 backup of the root dbspace before mirroring starts.

Chunks have status flags that indicate the following information:

- Whether the chunk is a primary or mirror chunk
- Whether the chunk is currently online, down, a new mirror chunk that requires a level-0 backup of the root dbspace, or in the process of being recovered

For descriptions of these chunk status flags, refer to the description of the **onstat -d** option in the *IBM Informix Administrator's Reference*. For information on how to display these status flags, refer to "Monitoring Disk Usage" on page 10-33.

Recovery

When the database server recovers a mirror chunk, it performs the same recovery procedure that it uses when mirroring begins. The mirror-recovery process consists of copying the data from the existing online chunk onto the new, repaired chunk until the two are identical.

When you initiate recovery, the database server puts the down chunk in recovery mode and copies the information from the online chunk to the recovery chunk. When the recovery is complete, the chunk automatically receives online status. You perform the same steps whether you are recovering the primary chunk of a mirrored pair or recovering the mirror chunk.

Tip: You can still use the online chunk during the recovery process. If data is written to a page that has already been copied to the recovery chunk, the database server updates the corresponding page on the recovery chunk before it continues with the recovery process.

For information on how to recover a down chunk, refer to the information on recovering a mirror chunk on page "Recovering a Mirror Chunk" on page 18-6.

Actions During Processing

This section discusses some of the details of disk I/O for mirror chunks and how the database server handles media failure for these chunks.

Disk Writes to Mirror Chunks

During database server processing, the database server performs mirroring by executing two parallel writes for each modification: one to the primary chunk and one to the mirror chunk.

Disk Reads from Mirror Chunks

The database server uses mirroring to improve read performance because two versions of the data reside on separate disks. A data page is read from either the primary chunk or the mirror chunk, depending on which half of the chunk includes the address of the data page. This feature is called a *split read*. Split reads improve performance by reducing the disk-seek time. Disk-seek time is reduced because the maximum distance over which the disk head must travel is reduced by half. Figure 17-2 illustrates a split read.

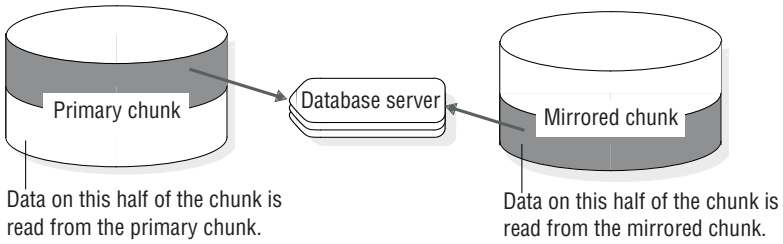


Figure 17-2. Split Read Reducing the Maximum Distance Over Which the Disk Head Must Travel

Detection of Media Failures

The database server checks the return code when it first opens a chunk and after any read or write. Whenever the database server detects that a primary (or mirror) chunk device has failed, it sets the chunk-status flag to down (D). For information on chunk-status flags, refer to "Mirror Status Flags" on page 17-4.

If the database server detects that a primary (or mirror) chunk device has failed, reads and writes continue for the one chunk that remains online. This statement is true even if the administrator intentionally brings down one of the chunks.

After the administrator recovers the down chunk and returns it to online status, reads are again split between the primary and mirror chunks, and writes are made to both chunks.

Chunk Recovery

The database server uses asynchronous I/O to minimize the time required for recovering a chunk. The read from the chunk that is online can overlap with the write to the down chunk, instead of the two processes occurring serially. That is, the thread that performs the read does not have to wait until the thread that performs the write has finished before it reads more data.

Result of Stopping Mirroring

When you end mirroring, the database server immediately frees the mirror chunks and makes the space available for reallocation. The action of ending mirroring takes only a few seconds.

Create a level-0 backup of the root dbspace after you end mirroring to ensure that the reserved pages with the updated mirror-chunk information are copied to the backup. This action prevents the restore procedure from assuming that mirrored data is still available.

Structure of a Mirror Chunk

The mirror chunk contains the same control structures as the primary chunk, as follows:

- Mirrors of blobspace chunks contain blobspace overhead pages.
- Mirrors of dbspace chunks contain dbspace overhead pages.
- Mirrors of sbspaces contain metadata pages.

For information on these structures, refer to the section on the structure of a mirror chunk in the disk structures and storage chapter of the *IBM Informix Administrator's Reference*.

A display of disk-space use, provided by one of the methods discussed under "Monitoring Chunks" on page 10-34, always indicates that the mirror chunk is full, even if the primary chunk has free space. The *full* mirror chunk indicates that none of the space in the chunk is available for use other than as a mirror of the primary chunk. The status remains full for as long as both primary chunk and mirror chunk are online.

If the primary chunk goes down and the mirror chunk becomes the primary chunk, disk-space allocation reports then accurately describe the fullness of the new primary chunk.

Chapter 18. Using Mirroring

In This Chapter

This chapter describes the various mirroring tasks that are required to use the database server mirroring feature. It provides an overview of the steps required for mirroring data.

Preparing to Mirror Data

This section describes how to start mirroring data on a database server that is not running with the mirroring function enabled.

To prepare to mirror data

1. Take the database server offline and enable mirroring.

   See "Enabling the MIRROR Configuration Parameter."
2. Bring the database server back online.
3. Allocate disk space for the mirror chunks.

   You can allocate this disk space at any time, as long as the disk space is available when you specify mirror chunks in the next step. The mirror chunks should be on a different disk than the corresponding primary chunks. See "Allocating Disk Space for Mirrored Data" on page 18-2.
4. Choose the dbspace, blobspace, or sbspace that you want to mirror, and specify a mirror-chunk path name and offset for each primary chunk in that storage space.

   The mirroring process starts after you perform this step. Repeat this step for all the storage spaces that you want to mirror. See "Using Mirroring" on page 18-3.

Enabling the MIRROR Configuration Parameter

Enabling mirroring invokes the database server functionality required for mirroring tasks. However, when you enable mirroring, you do not initiate the mirroring process. Mirroring does not actually start until you create mirror chunks for a dbspace, blobspace, or sbspace. See "Using Mirroring" on page 18-3.

Enable mirroring when you initialize the database server if you plan to create a mirror for the root dbspace as part of initialization; otherwise, leave mirroring disabled. If you later decide to mirror a storage space, you can change the value of the MIRROR configuration parameter.

To enable mirroring for the database server, you must set the MIRROR parameter in **ONCONFIG** to 1. The default value of MIRROR is `0`, indicating that mirroring is disabled.

Do not set the MIRROR parameter to 1 if you are not using mirroring.

To change the value of MIRROR, you can edit the **ONCONFIG** file with a text editor or ISA while the database server is in online mode. After you change the **ONCONFIG** file, take the database server offline and then to quiescent for the change to take effect.

Changing the MIRROR Parameter with ON-Monitor (UNIX)

To enable mirroring, choose **Parameters > Initialize**. In the field that is labelled **Mirror**, enter `Y`. Press `ESC` to record changes.

After the last of these screens, a prompt appears to confirm that you want to continue (to initialize the database server disk space and destroy all existing data). Respond `N` (no) to this prompt.

Warning: If you respond *Y* (yes) at this prompt, you lose all your existing data.

Take the database server offline and then to quiescent mode for the change to take effect.

Allocating Disk Space for Mirrored Data

Before you can create a mirror chunk, you must allocate disk space for this purpose. You can allocate either raw disk space or cooked file space for mirror chunks. For a discussion of allocating disk space, refer to "Allocating Disk Space" on page 10-1.

Always allocate disk space for a mirror chunk on a different disk than the corresponding primary chunk with, ideally, a different controller. This setup allows you to access the mirror chunk if the disk on which the primary chunk is located goes down, or vice versa.

Linking Chunks (UNIX)

Use the UNIX link (**ln**) command to link the actual files or raw devices of the mirror chunks to mirror path names. If a disk failure occurs, you can link a new file or raw device to the path name, eliminating the need to physically replace the disk that failed before the chunk is brought back online.

Relinking a Chunk to a Device After a Disk Failure

On UNIX, if the disk on which the actual mirror file or raw device is located goes down, you can relink the chunk to a file or raw device on a different disk. This action allows you to recover the mirror chunk before the disk that failed is brought back online. Typical

UNIX commands that you can use for relinking are shown in the following examples.

The original setup consists of a primary root chunk and a mirror root chunk, which are linked to the actual raw disk devices, as follows:

```
ln -l
lrwxrwxrwx 1 informix 10 May 3 13:38 /dev/root@->/dev/rxy0h
lrwxrwxrwx 1 informix 10 May 3 13:40 /dev/mirror_root@->/dev/rsd2b
```

Assume that the disk on which the raw device **/dev/rsd2b** resides has gone down. You can use the **rm** command to remove the corresponding symbolic link, as follows:

```
rm /dev/mirror_root
```

Now you can relink the mirror chunk path name to a raw disk device, on a disk that is running, and proceed to recover the chunk, as follows:

```
ln -s /dev/rab0a /dev/mirror_root
```

Using Mirroring

Mirroring starts when you create a mirror chunk for each primary chunk in a dbspace, blobspace, or sbspace.

When you create a mirror chunk, the database server copies data from the primary chunk to the mirror chunk. When this process is complete, the database server begins mirroring data. If the primary chunk contains logical-log files, the database server does not copy the data immediately after you create the mirror chunk but waits until you perform a level-0 backup. For an explanation of this behavior see "Creation of a Mirror Chunk" on page 17-3.

Important: You must always start mirroring for an entire dbspace, blobspace, or sbspace. The database server does not permit you to select particular chunks in a dbspace, blobspace, or sbspace to mirror. You must create mirror chunks for every chunk in the space.

You start mirroring a storage space when you perform the following operations:

- Create a mirrored root dbspace during system initialization
- Change the status of a dbspace from unmirrored to mirrored
- Create a mirrored dbspace, blobspace, or sbspace

Each of these operations requires you to create mirror chunks for the existing chunks in the storage space.

Mirroring the Root Dbspace During Initialization

If you enable mirroring when you initialize the database server, you can also specify a mirror path name and offset for the root chunk. The database server creates the mirror chunk when the server is initialized. However, because the root chunk contains logical-log files, mirroring does not actually start until you perform a level-0 backup.

To specify the root mirror path name and offset, set the values of MIRRORPATH and MIRROROFFSET in the ONCONFIG file before you bring up the database server.

If you do not provide a mirror path name and offset, but you do want to start mirroring the root dbspace, you must change the mirroring status of the root dbspace after the database server is initialized.

Changing the Mirror Status

You can make the following two changes to the status of a mirror chunk:

- Change a mirror chunk from online to down
- Change a mirror chunk from down to recovery

You can take down or restore a chunk only if it is part of a mirrored pair. You can take down either the primary chunk or the mirror chunk, as long as the other chunk in the pair is online.

For information on how to determine the status of a chunk, refer to "Monitoring Disk Usage" on page 10-33.

Managing Mirroring

You can use the **onspaces** utility to manage mirroring. On UNIX, you can also use ON–Monitor to manage mirroring. For a full description of the **onspaces** syntax, see information on the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Starting Mirroring for Unmirrored Storage Spaces

You can prepare mirroring for a dbspace, blobspace, or sbspace at any time. However, the mirroring does not start until you perform a level-0 backup.

Starting Mirroring for Unmirrored dbspaces Using onspaces

You can use the **onspaces** utility to start mirroring a dbspace, blobspace, or sbspace. For example, the following **onspaces** command starts mirroring for the dbspace **db_project**, which contains two chunks, **data1** and **data2**:

```
onspaces -m db_project\
-p /dev/data1 -o 0 -m /dev/mirror_data1 0\
-p /dev/data2 -o 5000 -m /dev/mirror_data2 5000
```

The following example shows how to turn on mirroring for a dbspace called **sp1**. You can either specify the primary path, primary offset, mirror path, and mirror offset in the command or in a file.

```
onspaces -m sp1 -f mirfile
```

The **mirfile** file contains the following line:

```
/ix/9.3/sp1 0 /ix/9.2/sp1mir 0
```

In this line, **/ix/9.3/sp1** is the primary path, `0` is the primary offset, **/ix/9.3/sp1mir** is the mirror path, and `0` is the mirror offset.

Starting Mirroring Using ISA

To start mirroring with ISA:

1. Select **Storage > Chunks**.
2. Select the dbspace name and click **Start Mirroring**.

Starting Mirroring for Unmirrored dbspaces Using ON-Monitor (UNIX)

Use the **Dbspaces > Mirror** option to start mirroring a dbspace.

To select the dbspace that you want to mirror, move the cursor down the list to the correct dbspace and press `CTRL-B`. The **Mirror** option then displays a screen for each chunk in the dbspace. You can enter a **mirror** path name and offset in this screen. After you enter information for each chunk, press `ESC` to exit the option. The database server recovers the new mirror chunks, meaning it copies data from the primary to the mirror chunk. If the chunk contains logical-log files, recovery is postponed until after you create a level-0 backup.

Starting Mirroring for New Storage Spaces

You can also start mirroring when you create a new dbspace, blobspace, or sbspace.

Starting Mirroring for New Spaces Using onspaces

You can use the **onspaces** utility to create a mirrored dbspace. For example, the following command creates the dbspace **db_acct** with an initial chunk **/dev/chunk1** and a mirror chunk **/dev/mirror_chk1**:

```
onspaces -c -d db_acct -p /dev/chunk1 -o 0 -s 2500 -m /dev/mirror_chk1 0
```

Another way to start mirroring is to select **Index by Utility > onspaces > -m**.

Starting Mirroring for New Spaces Using ISA

To start mirroring for new storage spaces with ISA

1. Select **Storage > Spaces**.
2. Click **Add Dbspace**, **Add Blobspace**, or **Add Sbspace**.
3. Enter the path and offset for the mirror chunk.

Starting Mirroring for New dbspaces Using ON-Monitor (UNIX)

To create a dbspace with mirroring, choose the **Dbspaces > Create** option. This option displays a screen in which you can specify the path name, offset, and size of a primary chunk and the path name and offset of a mirror chunk for the new dbspace.

Adding Mirror Chunks

If you add a chunk to a dbspace, blobspace, or sbspace that is mirrored, you must also add a corresponding mirror chunk.

Adding Mirror Chunks Using onspaces

You can use the **onspaces** utility to add a primary chunk and its mirror chunk to a dbspace, blobspace, or sbspace. The following example adds a chunk, **chunk2**, to the **db_acct** dbspace. Because the dbspace is mirrored, a mirror chunk, **mirror_chk2**, is also added.

```
onspaces -a db_acct -p /dev/chunk2 -o 5000 -s 2500 -m /dev/mirror_chk2 5000
```

Adding Mirror Chunks Using ISA

To add mirror chunks with ISA:

1. Select **Storage > Chunks**.
2. Select the dbspace name and click **Add Chunk**.
3. Enter the path and offset for the mirror chunk.

Adding Mirror Chunks Using ON-Monitor (UNIX)

In ON–Monitor, the **Dbspaces > Add-chunk** option displays fields in which to enter the primary-chunk path name, offset, and size, and the mirror-chunk path name and offset.

Taking Down a Mirror Chunk

When a mirror chunk is *down*, the database server cannot write to it or read from it. You might take down a mirror chunk to relink the chunk to a different device. (See "Relinking a Chunk to a Device After a Disk Failure" on page 18-2.)

Taking down a chunk is not the same as ending mirroring. You end mirroring for a complete dbspace, which causes the database server to drop all the mirror chunks for that dbspace.

Taking Down Mirror Chunks Using onspaces

You can use the **onspaces** utility to take down a chunk. The following example takes down a chunk that is part of the dbspace **db_acct**:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -D
```

Taking Down Mirror Chunks Using ON-Monitor (UNIX)

To use ON–Monitor to take down a mirror chunk, choose the **Dbspaces > Status** option. With the cursor on the dbspace that contains the chunk that you want to take down, press `F3` or `CTRL-B`. The database server displays a screen that lists all the chunks in the dbspace. Move the cursor to the chunk that you want to take down and press `F3` or `CTRL-B` to change the status (take it down).

Recovering a Mirror Chunk

To begin mirroring the data in the chunk that is online, you must recover the down chunk.

Recovering a Mirror Chunk Using onspaces

You can use the **onspaces -s** utility to recover a down chunk. For example, to recover a chunk that has the path name **/dev/mirror_chk1** and an offset of 0 kilobytes, issue the following command:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -O
```

Recovering a Mirror Chunk Using ISA

To recover a mirror chunk with ISA

Select **Index by Utility > onspaces > -s**.

Recovering a Mirror Chunk Using ON-Monitor (UNIX)

To use ON–Monitor to recover a down chunk, choose the **Dbspaces > Status** option.

Ending Mirroring

When you end mirroring for a dbspace, blobspace, or sbspace, the database server immediately releases the mirror chunks of that space. These chunks are immediately available for reassignment to other storage spaces.

Only users **informix** and **root** on UNIX or members of the **Informix-Admin** group on Windows can end mirroring.

You cannot end mirroring if any of the primary chunks in the dbspace are down. The system can be in online mode when you end mirroring.

Ending Mirroring Using onspaces

You can end mirroring with the **onspaces** utility. For example, to end mirroring for the root dbspace, enter the following command:

```
onspaces -r rootdbs
```

Another way to end mirroring is to select **Index by Utility > onspaces > -r**.

Ending Mirroring Using ON-Monitor (UNIX)

To end mirroring for a dbspace or blobspace with ON–Monitor, select the **Dbspaces > Mirror** option. Select a dbspace or blobspace that is mirrored and press CTRL-B or F3.

Ending Mirroring Using ISA

To end mirroring with ISA:

1. Select **Storage > Chunks**.
2. Select the dbspace name and click **Stop Mirroring**.

Chapter 19. Data Replication Overview (Enterprise/Workgroup Editions)

In This Chapter

This chapter contains general information about data replication options, including High-Availability Data Replication (HDR) secondary server, Shared Disk (SD) secondary server, and Remote Standalone (RS) secondary server.

The chapter discusses the following topics:

- Data replication
- How data replication works
- Secondary Server data updates
- Data Replication Configuration Examples
- Redirection and Connectivity for Data-Replication Clients
- Designing Data Replication Group Clients

Data Replication

Data Replication refers to the process of representing database objects at more than one distinct site. Data replication provides several configuration options. Applications can access data contained on secondary servers using read-only mode, or they can update data on secondary servers (see "Update Data on Secondary Servers" on page 19-11).

Table 19-1. Data Replication Secondary Server Types

Server type	Description
High-availability data replication (HDR) secondary server	Provides synchronous data replication for Dynamic Server. Use an HDR secondary server if you require a hot standby. Configuring an HDR secondary server provides a way to maintain a backup copy of the entire database server that applications can access quickly in the event of a catastrophic failure of the primary server.
Shared-Disk (SD) secondary server	A server that shares disk space with a primary server. An SD secondary server does not maintain a copy of the physical database on its own disk space; rather, it shares disks with the primary server.
Remote Standalone (RS) secondary server	A server that is updated asynchronously from the primary server. RS secondary servers can be geographically distant from the primary server, serving as remote back-up servers in disaster-recovery scenarios. Each RS secondary server maintains a complete copy of the database, with updates transmitted asynchronously from the primary server over secure network connections.

One way of replicating data is simply to copy a database to a database server installed on a different computer. This copy allows reports to access the data without disturbing client applications that use the original database.

Using data replication, however, the database server implements nearly transparent updates of entire database servers. All the data that one database server manages is replicated and dynamically updated on one or more secondary database servers, often at a separate geographical location.

Server Modes

The following table shows server modes for data replication configurations.

Server Mode	Explanation
Standard Mode	Not part of a data replication system
Primary Mode	The primary mode of a data replication system. Data can be updated.
Secondary Mode	The secondary mode of a data replication system.

A database server running in standard mode or a database server running in primary or secondary mode can be in any database server operating mode, such as quiescent or online. For information on operating modes, see "Database Server Operating Modes" on page 4-7.

Type of Data Replicated

Data replication replicates data in dbspaces and sbspaces, but does not replicate data in blobspaces.

All built-in and extended data types are replicated to the secondary server. User-defined types (UDTs) must be logged and reside in a single database server. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server. For data stored in an sbspace to be replicated, the sbspace must be logged.

Data stored in operating system files or persistent external files or memory objects associated with user-defined routines are not replicated.

User-defined types, user-defined routines, and DataBlade modules have special installation and registration requirements. For instructions, see "How Data Initially Replicates" on page 19-6.

Replicating Smart Large Objects

You may receive one or more of the following error messages while working with updatable secondary servers:

- 12014
- 12015
- 12233

These errors generally indicate a problem with a smart blob file descriptor. These errors can be caused by any of the following conditions:

- A smart large object identifier is passed to another transaction or process before committing the transaction. Because all objects including smart large objects are

uncommitted until the transaction is committed, do not allow other transactions to use the smart large object. In particular, dirty reads can access locked smart large objects.

- Smart large objects are not closed after opening them. At the end of a transaction, all smart large objects should be closed on secondary servers, especially those that are created and then the transaction is rolled back. Leaving smart large object file descriptors open causes memory to remain allocated until the session terminates.
- Another process has deleted the smart blob on the primary server. Share locks are not automatically propagated from secondary servers to the primary server so a different secondary server may access a smart large object that has actually been deleted on the primary. These accesses will work until the either the log record containing the delete is replayed on the secondary server or the secondary server is updated by the primary server.

Primary and Secondary Database Servers

When you configure a set of database servers to use data replication, one database server is called the *primary* database server, and the others are called *secondary* database servers. (In this context, a database server that does not use data replication is referred to as a *standard* database server.) The secondary server can include any combination of the SD secondary, RS secondary, and HDR secondary servers.

As Figure 19-1 illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.

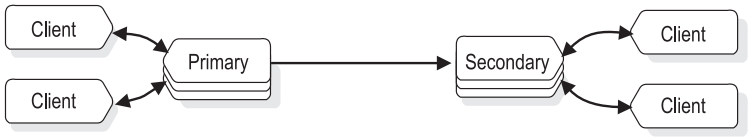


Figure 19-1. A Primary and Secondary Database Server in a Data Replication Configuration

If one of the database servers fails, as Figure 19-2 shows, you can redirect the clients that use that database server to the other database server in the pair, which becomes the primary server.

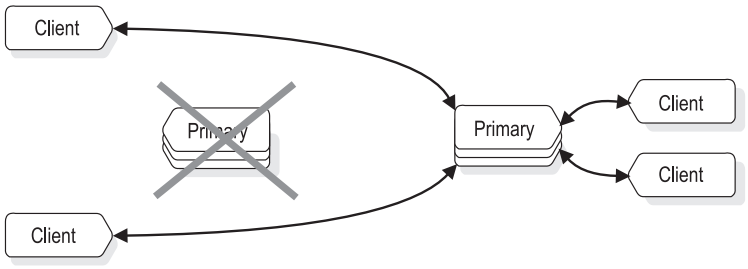


Figure 19-2. Database Servers and Clients in a Data Replication Configuration after a Failure

Advantages of Data Replication

Advantages of data replication are as follows:

- Clients at the site to which the data is replicated experience improved performance because those clients can access data locally rather than connecting to a remote database server over a network.
- Clients at all sites experience improved availability of replicated data. If the local copy of the replicated data is unavailable, clients can still access the remote copy of the data.

These advantages do not come without a cost. Data replication obviously requires more storage, and updating replicated data can take more processing time than updating a single object.

You could implement data replication in the logic of client applications by explicitly specifying where data must be updated. However, this method of achieving data replication is costly, prone to error, and difficult to maintain. Instead, the concept of data replication is often coupled with *replication transparency*. Replication transparency is built into a database server (instead of into client applications) to handle automatically the details of locating and maintaining data replicas.

Data Replication Versus Mirroring

Data replication and mirroring are both transparent ways of making the database server more fault tolerant. However, as Figure 19-3 on page 19-5, shows, they are quite different.

Mirroring, described "Mirroring" on page 17-1, is the mechanism by which a single database server maintains a copy of a specific dbspace on a separate disk. This mechanism protects the data in mirrored dbspaces against disk failure because the database server automatically updates data on both disks and automatically uses the other disk if one of the dbspaces fails.

Alternatively, data replication duplicates on an entirely separate database server all the data that a database server manages (not just specified dbspaces). Because data replication involves two separate database servers, it protects the data that these database servers manage, not just against disk failures, but against all types of database server failures, including a computer failure or the catastrophic failure of an entire site.

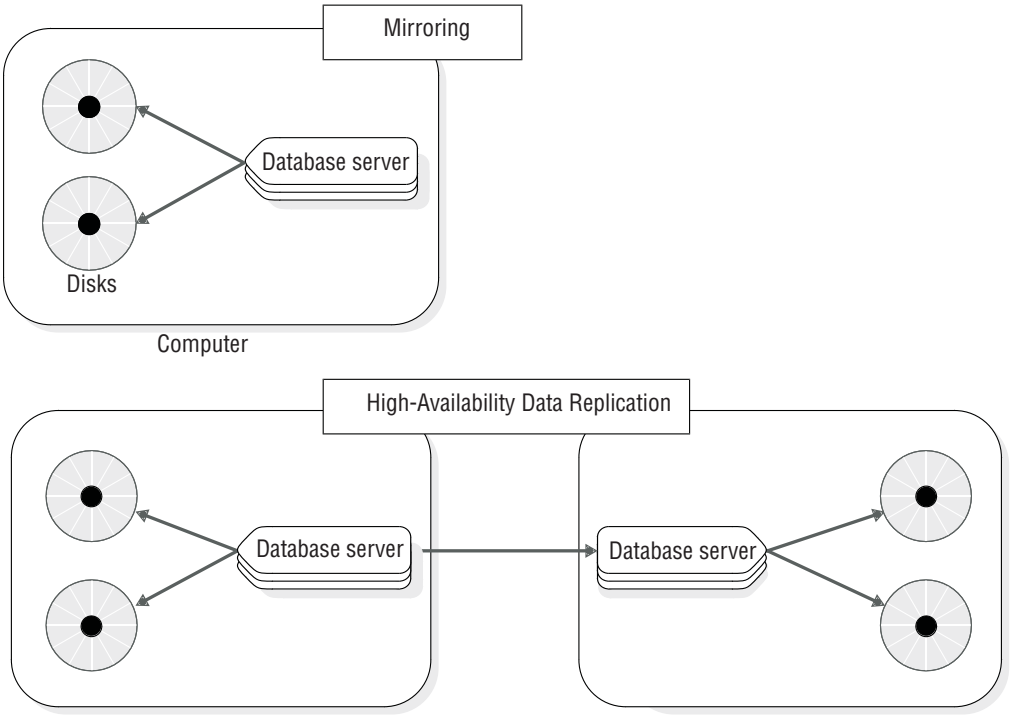


Figure 19-3. A Comparison of Mirroring and Data Replication

Data Replication Versus Two-Phase Commit

The two-phase commit protocol, described in detail in Chapter 25, "Multiphase Commit Protocols," on page 25-1, ensures that transactions are uniformly committed or rolled back across multiple database servers.

In theory, you could take advantage of two-phase commit to replicate data by configuring two database servers with identical data and then defining triggers on one of the database servers that replicate updates to the other database server. However, this sort of implementation has numerous synchronization problems in different failure scenarios. Also, the performance of distributed transactions is inferior to data replication.

Data Replication and Enterprise Replication

You can combine data replication with Enterprise Replication to create a robust replication system. Data replication can ensure that an Enterprise Replication system remains fully connected by providing backup database servers for critical replication nodes.

When you combine data replication and Enterprise Replication, only the primary server is connected to the Enterprise Replication system. No secondary servers participate in Enterprise Replication unless the primary server fails.

For more information, see *IBM Informix Dynamic Server Enterprise Replication Guide* and "Using Enterprise Replication as Part of the Recoverable Group" on page 19-21.

How Data Replication Works

This section describes the mechanisms that the database server uses to perform replication of data to secondary servers. For instructions on how to set up, start, and administer the various types of secondary servers, refer to Table 19-2

Table 19-2. Secondary server setup information

Secondary server type	See
HDR secondary	See Chapter 20, "Using High-Availability Data Replication (Enterprise/Workgroup Editions)," on page 20-1, and information on starting an HDR pair using external backup and restore in the *IBM Informix Backup and Restore Guide*.
RS secondary	See Chapter 21, "Using Remote Standalone Secondary Servers (Enterprise Edition)," on page 21-1.
SD secondary	See Chapter 22, "Using Shared Disk Secondary Servers (Enterprise Edition)," on page 22-1

How Data Initially Replicates

HDR secondary and RS secondary servers use storage-space backups and logical-log backups (both those backed up to tape and those on disk) to perform an initial replication of the data on the primary database server to the secondary database server.

SD secondary servers do not require a backup and restore from the primary server because SD secondary servers share the same disks as the primary.

To replicate data

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers.
2. Register user-defined types, user-defined routines, and DataBlade modules on the primary database server only.
3. To synchronize the data managed by the two database servers, create a level-0 backup of all the storage spaces on the primary database server.
4. Restore all the storage spaces from the backup on the secondary database server in the data-replication pair.

   The secondary database server that you restored from a storage-space backup in the previous step then reads all the logical-log records generated since that backup from the primary database server.

   The database server reads the logical-log records first from any backed-up logical-log files that are no longer on disk and then from any logical-log files on disk.

For detailed instructions about replicating data, refer to "Starting HDR for the First Time" on page 20-5. The *IBM Informix Backup and Restore Guide* discusses how to start replication using ON–Bar.

You must perform the initial backup with a storage-space backup. You cannot use data-migration utilities such as **onload** and **onunload** to replicate data because the physical page layout of tables on each database server must be identical in order for data replication to work.

Reproduction of Updates from the Primary Database Server

All secondary server types use logs to replicate data from the primary server. RS secondary servers require index page logging, but HDR secondary and SD secondary servers also use index page logging if it is enabled. For HDR secondary and RS secondary servers, updates made to the primary server are replicated on the secondary server by having the primary server send all its logical-log records to the secondary server as the logs are generated. HDR secondary and RS secondary servers receive the logical-log records generated on the primary server and apply them to their own dbspaces. For SD secondary servers, the primary server sends the log position of the logical log page to the SD secondary server. Using the log position received from the primary server, the SD secondary server reads the logical log page from disk and applies it to the memory data buffers.

Important: The database server cannot replicate updates to databases that do not use transaction logging.

How the Log Records Are Sent to HDR Secondary Servers

As shown in Figure 19-4 on page 19-8, when the primary database server starts to flush the contents of the logical-log buffer in shared memory to the logical log on disk, the database server also copies the contents of the logical-log buffer to a *data-replication buffer* on the primary database server. The primary database server then sends these logical-log records to the HDR secondary database server.

The HDR secondary database server receives the logical-log records from the primary database server into a shared-memory *reception buffer* (which the database server automatically adjusts to an appropriate size for the amount of data being sent). The secondary database server then applies the logical-log records through logical recovery.

How the Log Records Are Sent to RS Secondary and SD Secondary Servers

For RS secondary servers, the log pages can be cached as they are being flushed to disk, or they can be read directly from the log on disk. For SD secondary servers, the logs are not sent at all, but instead only the log position is sent.

HDR Data Replication Buffers

The data replication buffers are part of the virtual shared memory that the primary database server manages. The data replication buffers hold logical-log records before the primary database server sends them to the HDR secondary database server. The data replication buffers are the same size as the logical-log buffers. Figure 19-4 on page 19-8 shows this concept.

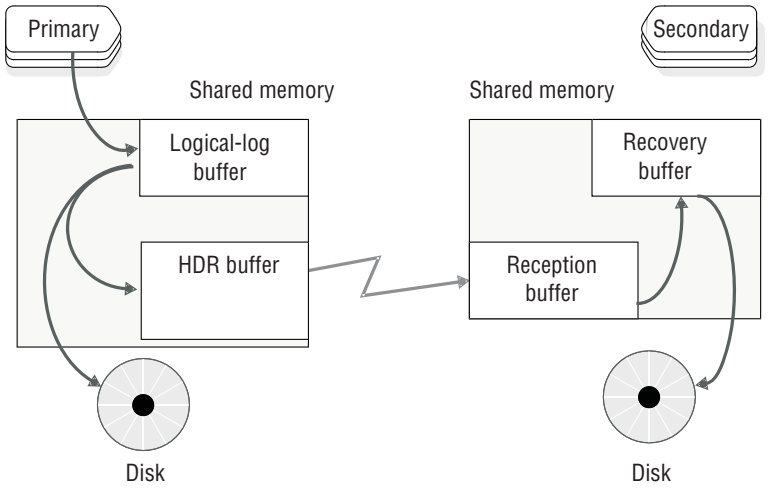


Figure 19-4. How Logical-Log Records Are Sent from the Primary Database Server to the Secondary Database Server

When Log Records Are Sent

The primary database server sends the contents of the data replication buffer to an HDR secondary server either *synchronously* or *asynchronously*. The value of the ONCONFIG configuration parameter DRINTERVAL determines whether the database server uses synchronous or asynchronous updating. For more information on DRINTERVAL, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

For RS secondary servers, index pages are sent asynchronously from the primary server using index page logging. For SD secondary servers, only the log page position within the logical log is replicated.

HDR Synchronous Updating

If you set DRINTERVAL to `-1`, data replication to the HDR secondary server occurs *synchronously*. As soon as the primary database server writes the logical-log buffer contents to the HDR buffer, it sends those records from the buffer to the HDR secondary database server. The logical-log buffer flush on the primary database server completes only after the primary database server receives acknowledgment from the HDR secondary database server that the records were received.

With synchronous updating, no transactions committed on the primary database server are left uncommitted or partially committed on the HDR secondary database server if a failure occurs.

HDR Asynchronous Updating

If you set DRINTERVAL to any value other than `-1`, data replication occurs *asynchronously* to the HDR secondary server. The primary database server flushes the logical-log buffer after it copies the logical-log buffer contents to the HDR buffer. Independent of that action, the primary database server sends the contents of the HDR buffer across the network when one of the following conditions occurs:

- The HDR buffer becomes full.
- The time interval, specified by the DRINTERVAL configuration parameter on the primary database server, has elapsed since the last time that records were sent to the secondary database server.

This method of updating might provide better performance than synchronous updating. However, as the following section explains, transactions can be lost.

Lost-and-Found Transactions: With asynchronous updating, a transaction committed on the primary database server might not be replicated on the secondary database server. This situation can result if a failure occurs after the primary database server copies a commit record to the HDR buffer but before the primary database server sends that commit record to the secondary database server.

If the secondary database server is changed to a standard database server after a failure of the primary database server, it rolls back any open transactions. These transactions include any that were committed on the primary database server but for which the secondary database server did not receive a commit record. As a result, transactions are committed on the primary database server but not on the secondary database server. When you restart data replication after the failure, the database server places all the logical-log records from the lost transactions in a file (which the DRLOSTFOUND configuration parameter specifies) during logical recovery of the primary database server. Figure 19-5 illustrates the process.

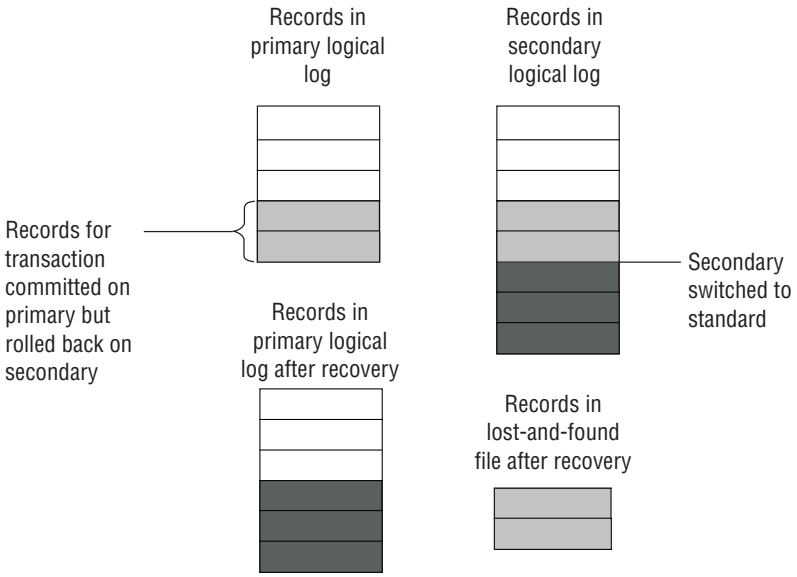


Figure 19-5. Using a Lost-and-Found File

If the lost-and-found file appears on the computer that is running the primary database server after it restarts data replication, a transaction has been lost. The database server cannot reapply the transaction records in the lost-and-found file because conflicting updates might have occurred while the secondary database server was acting as a standard database server.

To reduce the risk of a lost transaction without running data replication in synchronous mode, use unbuffered logging for all the databases. This method reduces the amount of time between the writing and transfer of the transaction records from the primary database server to the secondary database server.

Threads That Handle Data Replication

The primary database server starts specialized threads to support data replication. As Figure 19-6 shows, a thread called **drprsend** on the primary database server sends the contents of the primary server buffer across the network to a thread called **drsecrcv** on the secondary database server.

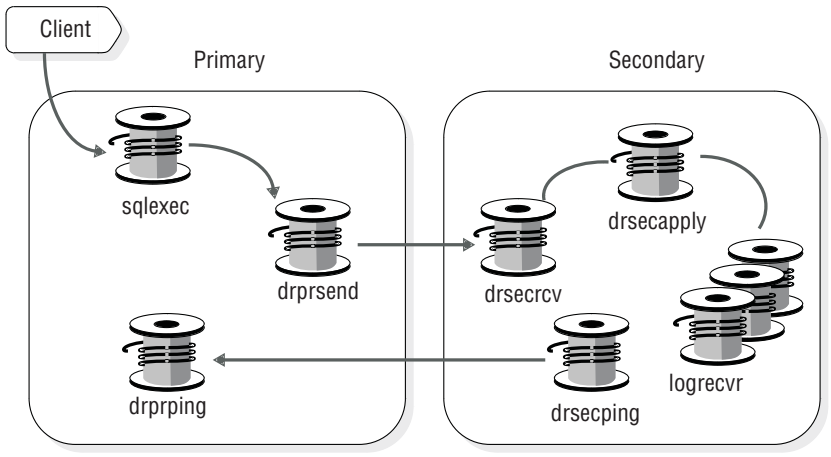


Figure 19-6. Threads That Manage Data Replication

A thread called **drsecapply** on the secondary database server copies the contents of the reception buffer to the recovery buffer. The **logrecvr** thread (or threads) performs logical recovery with the contents of the recovery buffer, applying the logical-log records to the dbspaces that the secondary database server manages. The OFF_RECVRY_THREADS configuration parameter specifies the number of **logrecvr** threads used.

The remaining threads that the database server starts are the **drprping** and **drsecping** threads, which are responsible for sending and receiving the messages that indicate if the two database servers are connected.

To support RS secondary servers, the primary server also checks to see whether there are RS secondary servers attached and if so, will copy the page to a log cache which is used to send that page to the RS secondary server. See Figure 19-7

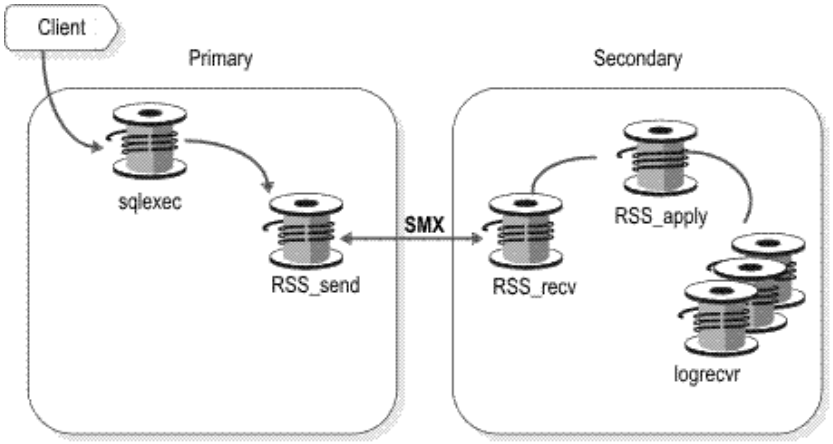


Figure 19-7. Threads that Manage Data Replication for RS Secondary Servers

The RSS_Send thread transmits the log page to the RS secondary server. It is possible that the next page that needs to be sent is not in the log cache. In that case, the RSS_Send thread will read the log pages directly from disk. The RSS_Send thread interfaces with SMX to send data in a fully duplexed manner. With full duplex communication, the thread does not wait for an acknowledgment from the RS secondary server before sending the next buffer. Up to 32 buffer transmissions will be sent before the primary server requires an acknowledgment from the RS secondary server. If the 32 buffer limit is reached, then the send thread will wait for the RSS_Recv thread to receive an acknowledgment from the RS secondary server.

On the RS secondary server, RSS_Recv interfaces with SMX to receive the log pages from the primary server.

Checkpoints Between Database Servers

Checkpoints between database servers in a replication pair are asynchronous. A checkpoint on the primary database server completes only after the checkpoint is started on the secondary database server. If the checkpoint does not complete within the time that the DRTIMEOUT configuration parameter specifies, the primary database server assumes that a failure has occurred. See "HDR Failures Defined" on page 20-20.

Tip: Synchronize the time on the operating systems of both database servers in the pair and set DRTIMEOUT on both servers to the same value.
Checkpoints between a primary server and an RS secondary server are asynchronous, and require index page logging to be enabled. See "Index Page Logging" on page 21-3.

How Data Synchronization Is Tracked

To keep track of synchronization, each database server in the pair keeps track of the following information in its reserved page:

- The ID of the logical-log file that contains the last completed checkpoint
- The position of the checkpoint record within the logical-log file
- The ID of the last logical-log file sent (or received)
- The page number of the last logical-log record sent (or received)

The database servers use this information internally to synchronize data replication.

Update Data on Secondary Servers

You can enable applications connected to secondary servers to update data. If you enable write operations on a secondary server, insert, update, and delete operations are propagated to the primary server. All types of secondary servers can be configured to perform write operations. Use the UPDATABLE_SECONDARY configuration parameter to control whether the secondary server can perform write operations and how many connections write operations can use.

Client applications can insert, update, and delete rows on a secondary server only if the secondary server image matches that of the primary server. This update capability is for any table that has all of its data in the table and lacks any blob data. The following data types are supported:

- BIGINT

- BIGSERIAL
- BLOB
- BOOLEAN
- BYTE
- CHAR
- CLOB
- DATE
- DECIMAL
- DTIME
- FLOAT
- INT
- INT8
- INTERVAL
- MONEY
- NCHAR
- NVCHAR
- SERIAL
- SERIAL8
- SMFLOAT
- SMINT
- TEXT
- VCHAR

The following data types are also supported if they do not receive a pointer reference to a different partition:

- COLLECTION
- LIST
- LVARCHAR
- MULTISET
- ROW
- SET
- UDTVAR

Any difference between the primary server image and the secondary server image causes an SQL error and the rollback of any changes.

You cannot use the following utilities on HDR secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers:

- archecker
- dbexport
- dbimport
- dbload
- HPL
- onaudit
- ondblog
- onload
- onmonitor

- onparams
- onperf
- onshowaudit
- onsnmp
- onspaces
- onunload

XA commands are not supported on secondary servers. In addition, SD secondary servers are not supported in Windows environments.

Byte range locking is not supported on secondary servers configured for updates. Byte range locks on secondary servers are promoted to full object locks.

Isolation Levels on Secondary Servers

The following statements are supported on all types of secondary servers:

```
Set isolation to committed read
Set isolation to committed read last committed
```

Secondary servers on which committed read isolation is set can read locally committed data. They can also read data committed on the primary server when it becomes available and committed on the secondary server. Applications connected to a secondary server receive data that is currently committed on the secondary server. See "Designing Data Replication Group Clients" on page 19-39 for additional information on design considerations for clients that connect to database servers that are running data replication.

The default isolation level on secondary servers is dirty read; however, setting an explicit isolation level enables the correct isolation level: dirty read, committed read, or committed read last committed.

Repeatable read and cursor stability isolation levels are not supported. Using the SET ISOLATION statement with cursor stability and repeatable read levels is ignored.

After starting a secondary server, client applications connect to the server only when all transactions open at the startup checkpoint have either committed or rolled back.

If the UPDATABLE_SECONDARY configuration parameter is disabled (by being unset or being set to zero), a secondary data replication server is read-only. In this case, only the DIRTY READ or READ UNCOMMITTED transaction isolation levels are available on HDR and RS secondary servers.

If the UPDATABLE_SECONDARY parameter is enabled (by setting it to a valid number of connections greater than zero), a secondary data replication server can support the COMMITTED READ, COMMITTED READ LAST COMMITTED, or COMMITTED READ transaction isolation level, or the USELASTCOMMITTED session environment variable. Only DML statements of SQL (the INSERT, UPDATE, and DELETE statements) can support write operations on an updatable secondary server.

Use **onstat -g ses** or **onstat -g sql** to view isolation level settings. See the *IBM Informix Dynamic Server Administrator's Reference* for more information.

Setting Lock Mode

Issuing a SET LOCK MODE TO WAIT or SET LOCK MODE TO WAIT *n* statement on a secondary server sets the lock wait timeout value for that session just like on a primary server. The value set by SET LOCK MODE will be used by the proxy thread created for the current session on the primary server when it performs updates from a secondary server. If the value for SET LOCK MODE is greater than the ONCONFIG parameter value of DEADLOCK_TIMEOUT, the value of DEADLOCK_TIMEOUT is used instead.

Transient Types on High-Availability Cluster Secondary Servers

Transient unnamed complex data types (ROW, SET, LIST, and MULTISET) can be used on high-availability cluster secondary servers, whether the secondary servers are read-only or updatable. The following types of operations that use transient types are supported on secondary servers:

- SQL queries that use transient types
- SQL queries that use derived tables, collection subqueries, and XML functions (these statements implicitly use transient types)
- Temporary tables created with the CREATE TEMP statement that use transient types

See the *IBM Informix Guide to SQL: Reference* and *IBM Informix Guide to SQL: Syntax* for information on complex data types.

Row Versioning

Use row versioning to determine whether a row was changed and to detect collisions. With row versioning enabled, each row of a table is configured to contain both a checksum and a version number. When a row is first inserted, the checksum is generated automatically, and the version is set to 1. Every time the row is updated the version is incremented by one, while the checksum value is not changed. With row versioning, if a row is deleted and another row is re-inserted in a table, it is possible to recognize that the row is different. By comparing the row checksum and row version between the secondary and the primary servers, it is possible to detect data collisions.

Web applications can use a version column to determine whether information contained in a previously retrieved object is still current. For example, a Web application might display items for sale to a customer. When the customer decides to purchase an item, the application can check the version column of the item's row to determine whether any information about the item has changed.

If client applications can update data on the secondary servers in your environment, use row versioning to minimize network use, especially if your tables have many columns. Otherwise, entire rows on the secondary server are compared with entire rows on the primary server to determine whether updates occurred.

To add row versioning to an existing table, use the following syntax:

```
ALTER TABLE tablename add VERCOLS;
```

Similarly, you can delete row versioning from a table with the following syntax:

```
ALTER TABLE tablename drop VERCOLS;
```

To create a new table with row versioning, use the following syntax:

```
CREATE TABLE tablename (
   Column Name   Datatype
   Column Name   Datatype
   Column Name   Datatype
   ) with VERCOLS;
```

Data Replication Configuration Examples

This section describes some examples of how a data replication environment can be configured.

Remote Standalone Secondary Configuration Examples

The following figure illustrates an example of a configuration consisting of multiple RS secondary servers. This configuration would be useful in a situation where the primary server is located a long distance from the RS secondary servers or if the network speed between the primary server and the RS secondary server is slow or erratic. Because RS secondary servers use fully duplexed communication protocols, and do not require checkpoints processed in SYNC mode, the additional servers should not have a significant impact on the performance of the primary server.

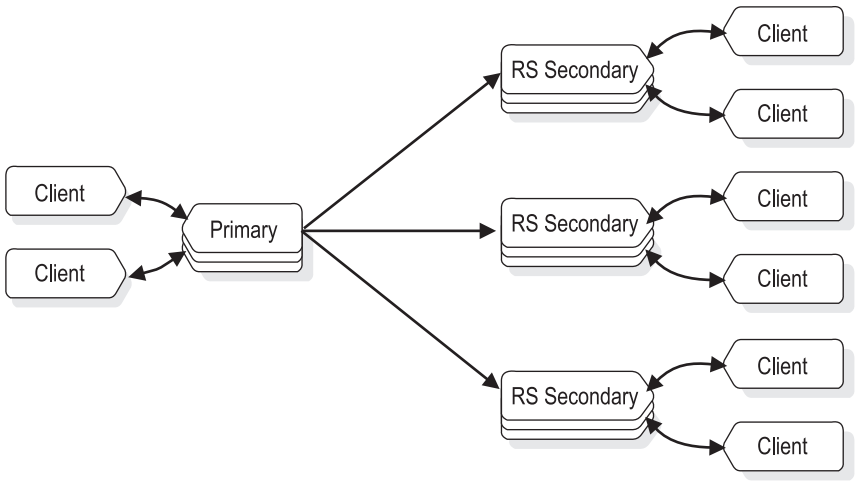


Figure 19-8. Primary Server with Three RS Secondary Servers

The next illustration shows an example of a configuration of an RS secondary server along with an HDR secondary server. In this example, the HDR secondary provides high availability while the RS secondary provides additional disaster recovery if both the primary and HDR secondary servers are lost. The RS secondary server can be geographically remote from the primary and HDR secondary servers so that a regional disruption such as an earthquake or flood would not affect the RS secondary server.

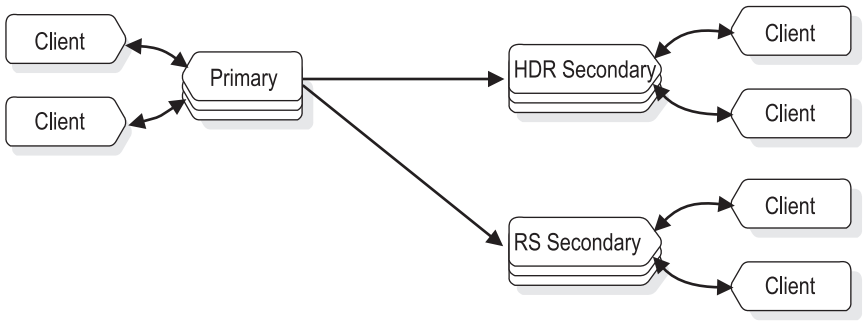


Figure 19-9. Primary Server with HDR Secondary and RS Secondary Servers

If a primary database server fails, it is possible to convert the existing HDR secondary server into the primary server, as in the following diagram:

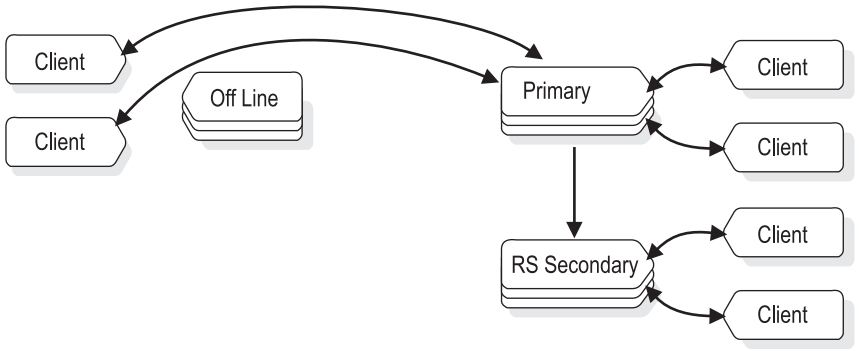


Figure 19-10. Failover of Primary Server to HDR Secondary Server

If it appears that the original primary is going to be off line for an extended period of time, then the RS secondary server can be converted to an HDR secondary server. Then when the original primary comes back on line, it can be configured as an RS secondary server, as in the following illustration:

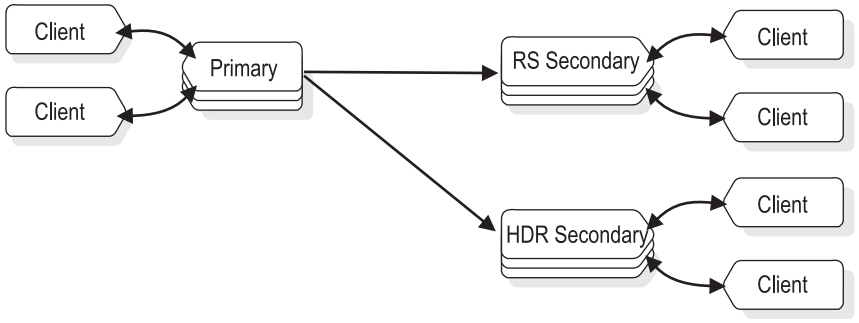


Figure 19-11. RS Secondary Server Assuming Role of HDR Secondary Server

Shared Disk Secondary Configuration Examples

The following figure shows an example of a primary server with two SD secondary servers. In this case the role of the primary server could be transferred

to either of the two SD secondary servers. This is true whether the primary needed to taken out of service because of a planned outage, or because of failure of the primary server.

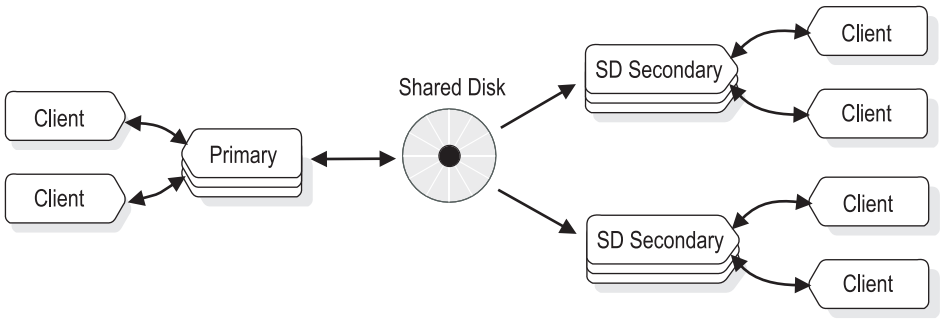


Figure 19-12. Primary Server Configured With Two SD Secondary Servers

Because both of the SD secondary servers are reading from the same disk subsystem, they are both equally able to assume the primary server role. The following figure illustrates a situation in which the primary server is offline.

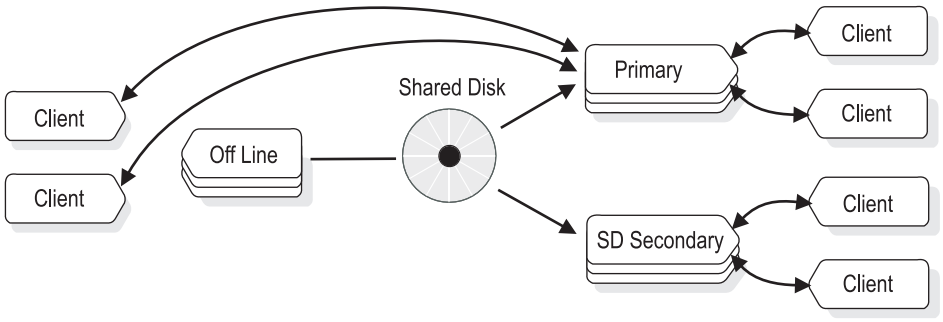


Figure 19-13. SD Secondary Server Assuming Role of Primary Server

There are several ways to protect against hardware failure of the shared disk. Probably the most common way is to configure the disk array based on RAID technology (such as RAID-5). Another way to protect against disk failure is to use SAN (Storage Area Technology) to include some form of remote disk mirroring. Since SAN disks can be located a short distance from the primary disk and its mirror, this provides a high degree of availability for both the planned and unplanned outage of either the server or of the disk subsystem. The following illustration depicts such a configuration:

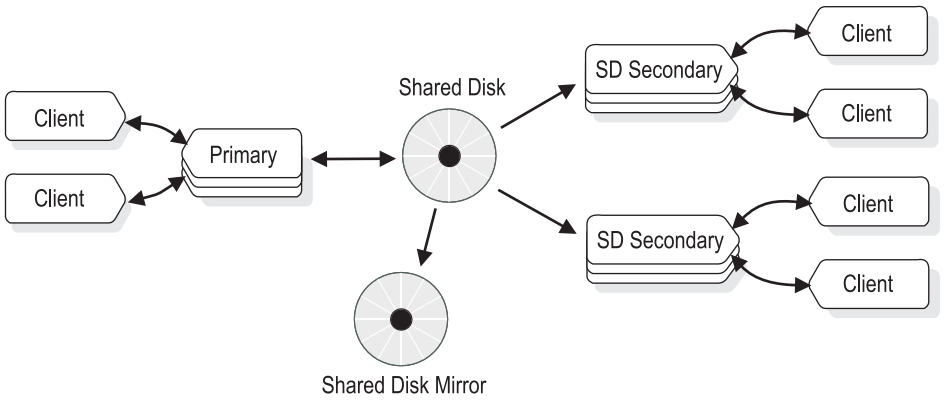


Figure 19-14. Primary Server and SD Secondary Servers with Mirrored Disks

In the event of a disk failure, the servers can be reconfigured as in the following illustration:

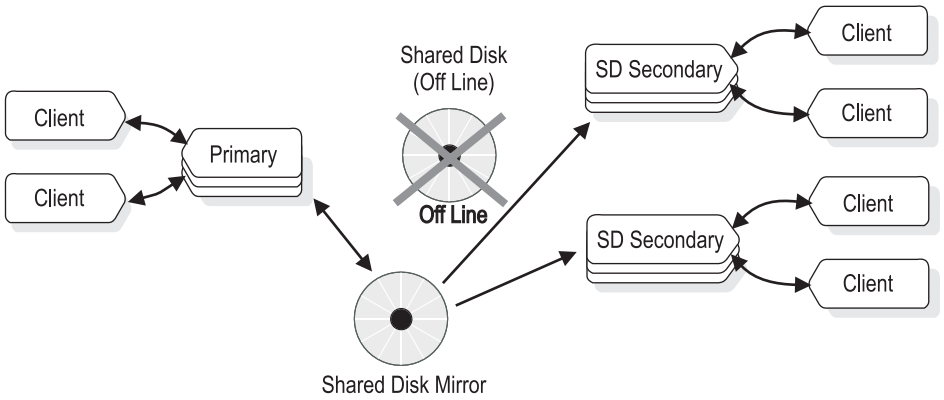


Figure 19-15. Shared Disk Mirror After Failure of Primary Shared Disk

In addition to configuring a mirrored disk subsystem as in the previous illustration, you might want to configure additional servers. For example, you might want to use the primary and two SD secondary servers within a single blade server enclosure. By placing the server group within a single blade server, the blade server itself can become a failure point. The configuration in the following illustration is an attractive solution when you need to periodically increase read processing ability such as when performing large reporting tasks.

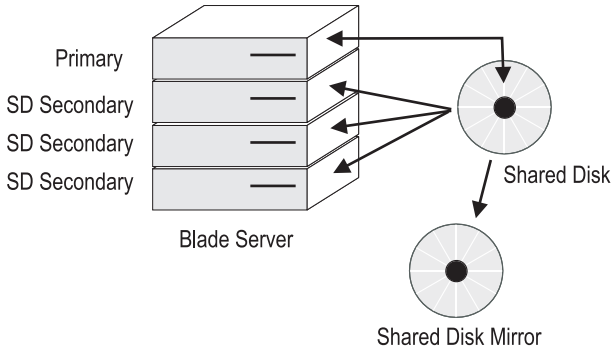


Figure 19-16. Primary and SD Secondary Servers in a Blade Server

You might decide to avoid the possible failure point of a single blade server by using multiple blade servers, as in the following illustration.

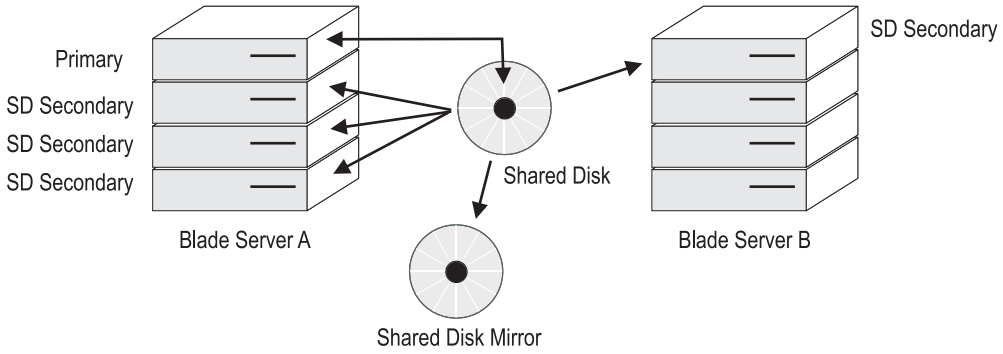


Figure 19-17. Multiple Blade Server Configuration to Prevent Single Point of Failure

In the previous illustration, if Blade Server A fails, it would be possible to transfer the primary server role to the SD secondary server on Blade Server B. Since it is possible to bring additional SD secondary servers online very quickly, it would be possible to dynamically add additional SD secondary servers to Blade Server B as in the following illustration.

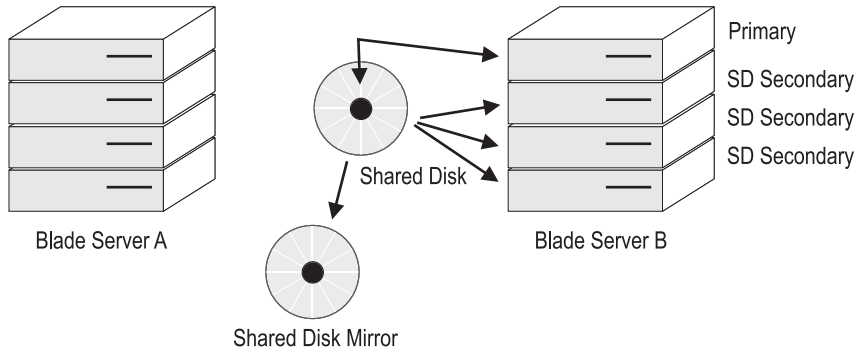


Figure 19-18. Failover After Failure of Blade Server

Because of limits on the distance between the primary and mirrored disks that disk mirroring can support, you might be concerned about using shared disks and relying on shared disk mirroring to provide disk availability. For example, you

might prefer that there be a significant distance between the two copies of the disk subsystem. In this case, you might choose to use either an HDR secondary or an RS secondary server to maintain the secondary copy of the disk subsystem. If the network connection is fairly fast (that is, if a ping to the secondary server is less than 50 milliseconds) you should consider using an HDR secondary server. For slower network connections, consider using an RS secondary server. The following illustration shows an example of an HDR secondary server in a blade server configuration.

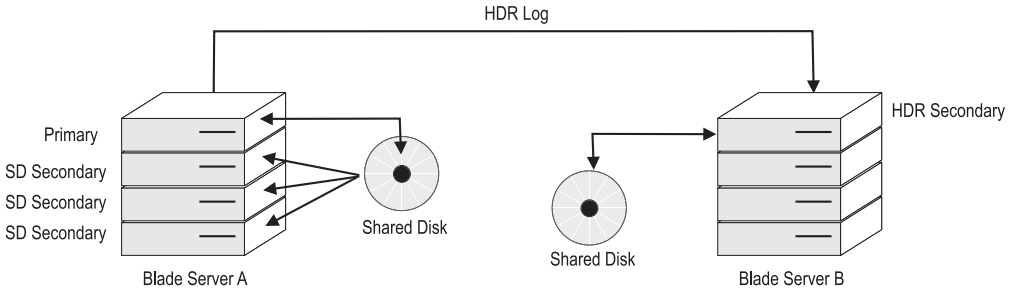


Figure 19-19. HDR Secondary Server in Blade Server Configuration

In the configuration shown in the previous illustration, if the primary node should fail, but the shared disks are intact and the blade server is still functional, it is possible to transfer the primary server role from the first server in Blade Server A to another server in the same blade server. Changing the primary server would cause the source of the remote HDR secondary server to automatically reroute to the new primary server, as illustrated in the following diagram:

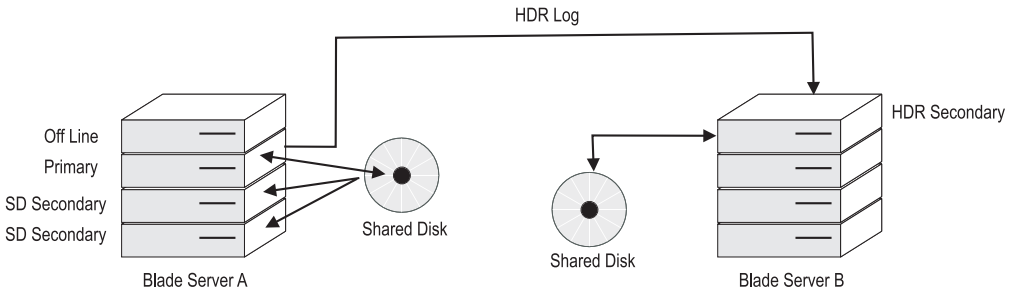


Figure 19-20. Failover of Primary Server to SD Secondary Server in Blade Server Configuration

Suppose, however, that the failure described in the previous illustration was not simply a blade within the blade server, but the entire blade server. In this case you might have to fail over to the HDR secondary. Since starting an SD secondary server is very quick, you could easily add additional SD secondary servers. Note that the SD secondary server can only work with the primary node; when the primary has been transferred to Blade Server B, then it becomes possible to start up SD secondary servers on Blade Server B as well, as shown in the following illustration.

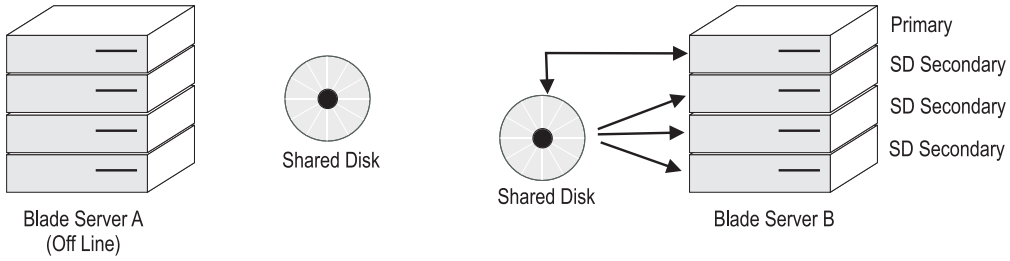


Figure 19-21. Failure of Entire Blade Server

Using Enterprise Replication as Part of the Recoverable Group

While Enterprise Replication does not support a SYNC (synchronous) mode of operation, it does provide the ability to support environments with multiple active servers. During a failover event, Enterprise Replication is able to reconcile database differences between two servers. You should consider Enterprise Replication as a means of improving synchronization between servers because each Enterprise Replication system maintains an independent logging system. A configuration using Enterprise Replication is shown in the following illustration:

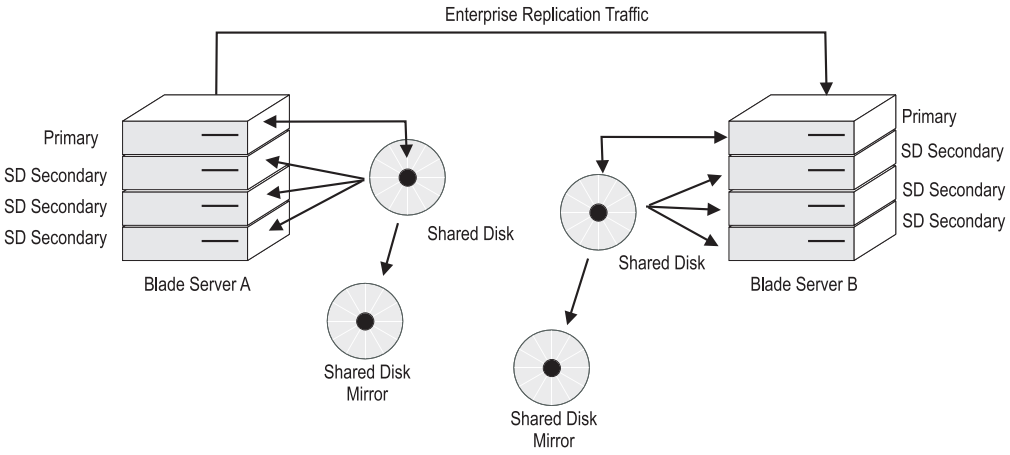


Figure 19-22. Configuring Enterprise Replication as Part of the Recoverable Group

High-Availability Clusters with Enterprise Replication Configuration Example

Suppose you need enterprise replication between two high-availability server clusters configured as follows:

Cluster 1:

- Primary server
- HDR secondary server
- SD secondary server 1
- SD secondary server 2
- RS secondary server 1
- RS secondary server 2

Cluster 2:

- Primary server
- HDR secondary server
- SD secondary server 1
- SD secondary server 2
- RS secondary server 1
- RS secondary server 2

Suppose further that each of the servers is named according to the following convention:

- First three characters: name of enterprise
- Character 4: host short number
- Characters 5, 6, and 7: cluster number
- Characters 8, 9, and 10: server type: ″pri″ for primary server, ″sec″ for secondary server
- Characters 11, 12, and 13: connection type: ″shm″ or ″tcp″

For example, a server with the name: **srv4_1_pri_shm** is described as follows:

- srv = name of enterprise
- 4 = host short number
- _1_ = cluster number
- pri = this is a primary server
- shm = connection type uses shared memory communication

The following entries in the **sqlhosts** file would support the above configuration:

```
srv4_1_pri_shm onipcshm sun-mach4 srv4_1_pri_shm
srv4_1_sec_shm onipcshm sun-mach4 srv4_1_sec_shm
srv5_1_rss_shm onipcshm sun-mach5 srv5_1_rss_shm
srv5_1_sds_shm onipcshm sun-mach5 srv5_1_sds_shm
srv6_1_rss_shm onipcshm sun-mach6 srv6_1_rss_shm
srv6_1_sds_shm onipcshm sun-mach6 srv6_1_sds_shm
srv_1_cluster group - - i=1
srv4_1_pri_tcp ontlitcp sun-mach4 21316 g=srv_1_cluster
srv4_1_sec_tcp ontlitcp sun-mach4 21317 g=srv_1_cluster
srv5_1_rss_tcp ontlitcp sun-mach5 21316 g=srv_1_cluster
srv5_1_sds_tcp ontlitcp sun-mach5 21317 g=srv_1_cluster
srv6_1_rss_tcp ontlitcp sun-mach6 21316 g=srv_1_cluster
srv6_1_sds_tcp ontlitcp sun-mach6 21317 g=srv_1_cluster

srv4_2_pri_shm onipcshm sun-mach4 srv4_2_pri_shm
srv4_2_sec_shm onipcshm sun-mach4 srv4_2_sec_shm
srv5_2_rss_shm onipcshm sun-mach5 srv5_2_rss_shm
srv5_2_sds_shm onipcshm sun-mach5 srv5_2_sds_shm
srv6_2_rss_shm onipcshm sun-mach6 srv6_2_rss_shm
srv6_2_sds_shm onipcshm sun-mach6 srv6_2_sds_shm
srv_2_cluster group - - i=2
srv4_2_pri_tcp ontlitcp sun-mach4 21318 g=srv_2_cluster
srv4_2_sec_tcp ontlitcp sun-mach4 21319 g=srv_2_cluster
srv5_2_rss_tcp ontlitcp sun-mach5 21318 g=srv_2_cluster
srv5_2_sds_tcp ontlitcp sun-mach5 21319 g=srv_2_cluster
srv6_2_rss_tcp ontlitcp sun-mach6 21318 g=srv_2_cluster
srv6_2_sds_tcp ontlitcp sun-mach6 21319 g=srv_2_cluster
```

Example of a Complex Failover Recovery Strategy

This section describes a three-tiered server approach for achieving maximum availability in the case of a large region-wide disaster.

In general, an HDR Secondary server provides backup for SD secondary servers and provides support for a highly available system which is geographically remote from the main system. RS secondary servers provide additional availability for the HDR secondary and should be viewed as a disaster availability solution. If you must use an RS secondary server for availability, then you will be forced to manually rebuild the other systems by performing backup and restore in order to return to normal operation. To further understand this, a scenario is presented in which a large region-wide disaster occurs, such as a hurricane.

To provide maximum availability to survive a regional disaster requires *layered* availability. The first layer provides availability solutions to deal with transitory local failures. For example, this might include having a couple of blade servers attached to a single disk subsystem running SD secondary servers. Placing the SD secondary servers in several locations throughout your campus makes it possible to provide seamless failover in the event of a local outage.

You might want to add a second layer to increase availability by including an alternate location with its own copy of the disks. To protect against a large regional disaster, you might consider configuring an HDR secondary server located some distance away, perhaps hundreds of miles. You might also want to make the remote system a blade server or some other multiple-server system. By providing this second layer, if a fail-over should occur and the remote HDR secondary became the primary, then it would be possible to easily start up SD secondary servers at the remote site.

However, even a two-tiered approach might not be enough. A hurricane in one region can spawn tornadoes hundreds of miles away. To protect against this, consider adding a third tier of protection, such as an RS secondary server located one or more thousand miles away. This three-tier approach provides for additional redundancy that can significantly reduce the risk of an outage.

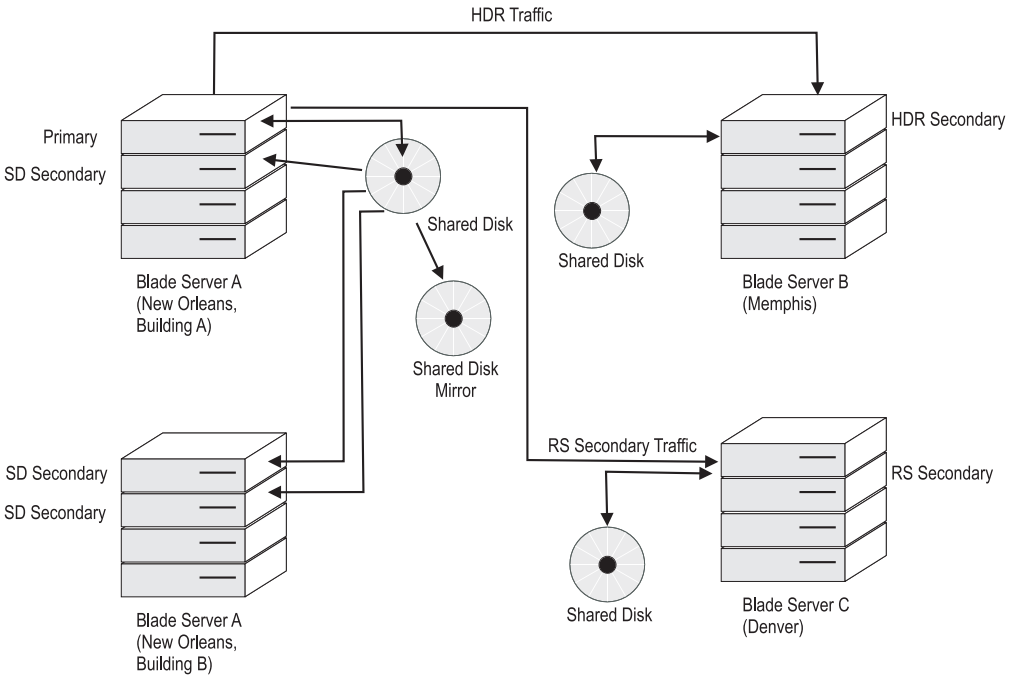


Figure 19-23. Configuration for Three-Tiered Server Availability

Now suppose that a local outage occurred in Building-A on the New Orleans campus. Perhaps a pipe burst in the machine room causing water damage to the blade server and the primary copy of the shared disk subsystem. You can switch the role of primary server to Building-B by running onmode -d make primary servername on one of the SD secondary servers running on the blade server in Building-B. This would cause all other secondary nodes to automatically connect to the new primary node.

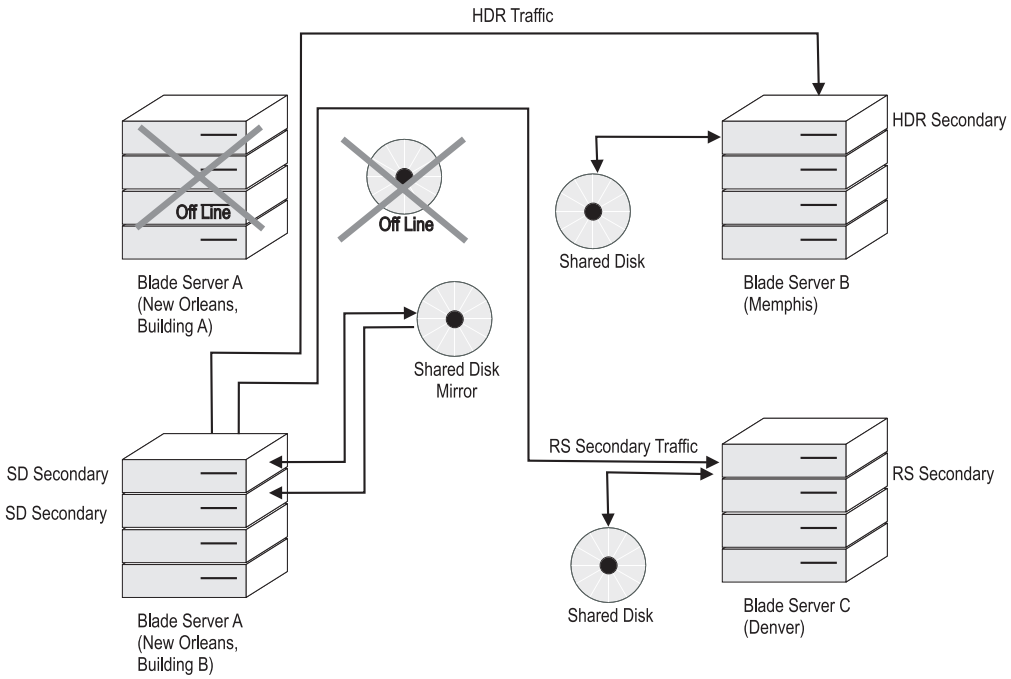


Figure 19-24. First Tier of Protection

Should there be a regional outage in New Orleans such that both building A and building B were lost, then you could shift the primary server role to Memphis. In addition, you might also want to make Denver into an HDR secondary and possibly add additional SD secondary servers to the machine in Memphis.

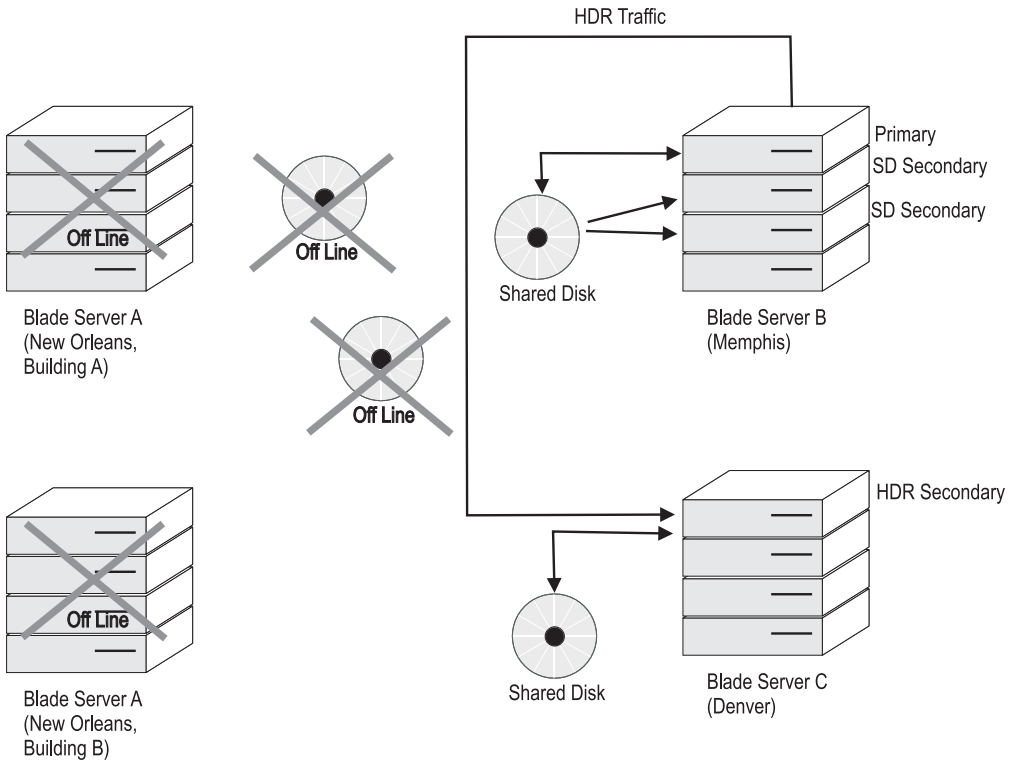


Figure 19-25. Second Tier of Protection

An even larger outage which affected both sites would require switching to the most remote system.

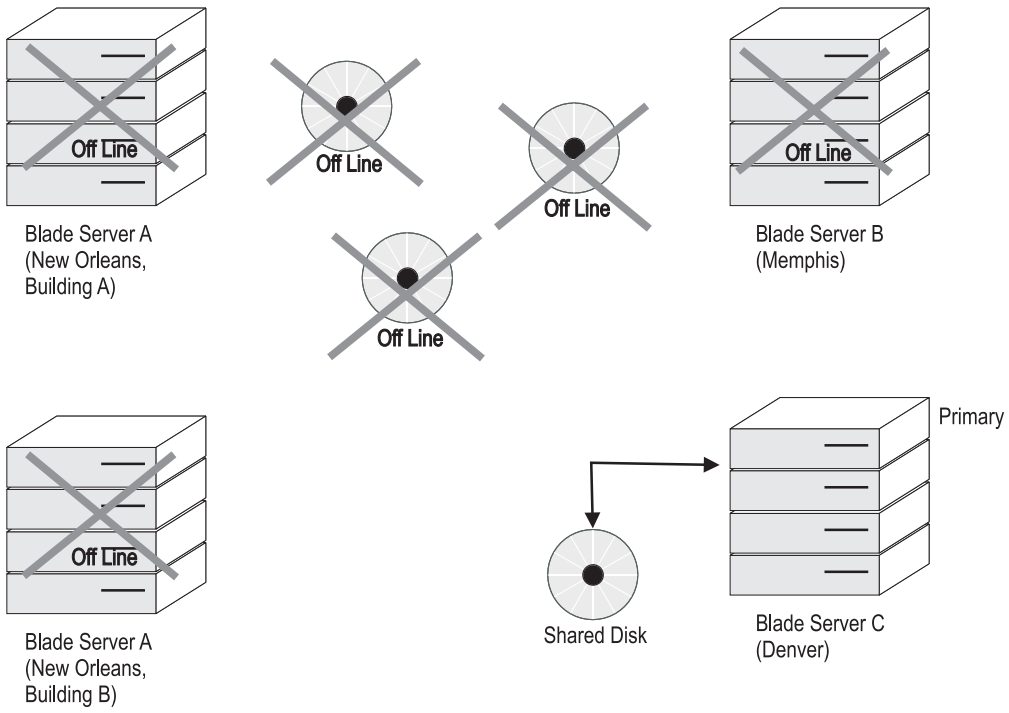


Figure 19-26. Third Tier of Protection

Table 19-3. Suggested Configurations for Various Requirements

Requirement	Suggested configuration
You periodically need to increase reporting capacity	Use SD secondary servers
You are using SAN devices, which provide ample disk hardware availability, but are concerned about server failures	Use SD secondary servers
You are using SAN devices, which provide ample disk hardware mirroring, but also want a second set of servers that are able to be brought online if the main operation should be lost (and the limitations of mirrored disks are not a problem)	Consider using two blade centers running SD secondary servers at the two sites
You want to have a backup site some moderate distance away, but can not tolerate any loss of data during failover	Consider using two blade centers with SD secondary servers on the main blade center and an HDR secondary on the remote.
You want to have a highly available system in which no transaction is ever lost, but must also have a remote system on the other side of the world	Consider using an HDR secondary located nearby running SYNC mode and an RS secondary server on the other side of the world
You want to have a high availability solution, but because of the networks in your region, the best response time from a ping is about 200 ms	Consider using an RS secondary server
You want a backup site but you do not have any direct communication with the backup site	Consider using Continuous Log Restore with backup and recovery
You can tolerate a delay in the delivery of data as long as the data arrives eventually; however you need to have quick failover in any case	Consider using SD secondary servers with hardware disk mirroring in conjunction with ER.
You need additional write processing power, can tolerate some delay in the delivery of those writes, need something highly available, and can partition the workload	Consider using ER with SD secondary servers

Troubleshooting High-Availability Cluster Environments

A high-availability cluster environment requires little or no additional troubleshooting than a standalone server environment. This section discusses the terminology used to describe high-availability cluster environments and provides some common troubleshooting procedures.

You use the diagnostic tools to display the configuration of a high-availability environment and to verify that your secondary servers are set up correctly to update data.

Transactions are processed by the servers very quickly. The onstat commands display status information only for the instant the command was run.

To update data on secondary servers, IDS creates *proxy distributors* on both the primary and the secondary database servers. Each proxy distributor is assigned an ID that is unique within the cluster. The proxy distributor is responsible for

sending DML update requests from secondary servers to the primary server. Secondary servers determine how many instances of the proxy distributors to create based on the UPDATABLE_SECONDARY setting in the secondary server's ONCONFIG file.

For updatable secondary servers in a high-availability cluster environment, encryption from the updatable secondary server to primary server requires SMX encryption. To encrypt data sent from an updatable secondary server to the primary server, set the **ENCRYPT_SMX** configuration parameter on the secondary server. See "Enabling SMX Encryption" on page 21-4 for more information.

Use the **onstat -g proxy** command on the primary server to view information about all proxy distributors in the high-availability cluster.

```
onstat -g proxy

IBM Informix Dynamic Server Version 11.50.U       -- On-Line -- Up 00:07:55 -- 46392 Kbytes
Secondary    Proxy      Transaction  Hot Row
Node         ID         Count        Total
serv2        392        2            112
serv2        393        2            150
```

Examining the output from the **onstat** command in the previous example, there are two proxy distributors whose IDs are 392 and 393. The Transaction Count indicates the number of transactions currently being processed by each proxy distributor.

You run **onstat -g proxy** on a secondary server to view information about the proxy distributors that are able to service update requests from the secondary server.

```
onstat -g proxy

IBM Informix Dynamic Server Version 11.50.U -- Updatable (SDS) -- Up 00:38:30 -- 62776 Kbytes
Primary            Proxy     Transaction  Hot Row
Node               ID        Count        Total
serv1              392       2            112
serv1              393       2            150
```

In this example, the server is a shared disk (SD) secondary server, and is configured to update data. In addition, the server is connected to the primary server named **serv1**, and there are two proxy distributors, each with a transaction count of 2.

Use **onstat -g proxy all** on the primary server to display information about proxy distributors and *proxy agent threads*. One or more proxy agent threads are created by the proxy distributor to handle data updates from the secondary server.

```
onstat -g proxy all

IBM Informix Dynamic Server Version 11.50.U       -- On-Line -- Up 00:08:28 -- 46392 Kbytes
Secondary          Proxy      Transaction  Hot Row
Node               ID         Count        Total
serv2              392        2            1
serv2              393        2            0

TID      Flags      Proxy  Source   Proxy   Current  sqlerrno iserrno
                    ID     SessID   TxnID   Seq
63       0x00000024 392    22       1        5        0        0
64       0x00000024 392    19       2        5        0        0
62       0x00000024 393    23       1        5        0        0
65       0x00000024 393    21       2        5        0        0
```

In the output of the **onstat -g proxy all** command, **TID** represents the ID of the proxy agent thread that is running on the primary server. **Source SessID** represents the ID of the user's session on the secondary server. **Proxy TxnID** displays the sequence number of the current transaction. Each **Proxy TxnID** is unique to the proxy distributor. **Current Seq** represents the sequence number of the current *operation* in the transaction being processed. Each database transaction sent to a secondary server is separated internally into one or more *operations* that are then sent to the primary server. The last two fields, **sqlerrno** and **iserrno**, display any SQL or ISAM/RSAM errors encountered in the transaction. An error number of 0 indicates completion with no errors.

Running **onstat -g proxy all** on the secondary server displays information about all of the sessions that are currently able to update data on secondary servers.

```
onstat -g proxy all

IBM Informix Dynamic Server Version 11.50.U -- Updatable (SDS) -- Up 00:13:34 -- 62776 Kbytes
Primary            Proxy      Transaction  Hot Row
Node               ID         Count        Total
serv1              3466       0            1
serv1              3465       1            0

Session  Proxy    Proxy    Proxy    Current  Pending  Reference
         ID       TID      TxnID    Seq      Ops      Count
19       3465     67       1        23       0        1
```

In the output from the **onstat -g proxy all** command run on the secondary server, **Session** represents the ID of a user's session on the secondary server. The **Proxy ID** and **Proxy TID** are the same as those displayed on the primary server. **Pending Ops** displays the number of operations that are waiting to be transferred to the primary server. **Reference Count** displays the number of threads in use for the transaction. When **Reference Count** displays 0 the transaction processing is complete.

To display detailed information about the current work being performed by a given distributor, use:

```
onstat -g proxy <proxy id> [proxy transaction id] [operation number]
```

The proxy transaction ID and operation number are both optional parameters. When supplied, the first number will be considered the proxy transaction ID. If a secondary number follows it will be interpreted as the operation number. If no proxy transaction ID exists, the command performs the same as:

```
onstat -
```

Similarly, if no such operation number for the given proxy transaction ID exists, the command performs the same as:

```
onstat -
```

Use the following command to display information about whether a server is configured to allow updates to data. The command can be run either on the primary or secondary server:

```
onstat -g <server_type>
```

Examples:

```
onstat -g rss
onstat -g sds
onstat -g dri
```

Configuring Connection Manager using Settings from the Primary Server

It is often useful to configure a Connection Manager instance using the same configuration parameters as those on the primary server. This is useful if you have existing applications that point to a primary server and you do not wish to recompile those applications to point to Connection Manager. Using the procedure that follows, you will configure a Connection Manager instance so that existing applications connect to Connection Manager instead of the primary server.

For this example, assume that a Connection Manager instance has not been configured, and you have a primary database server, an HDR secondary server, an SD secondary server, and an RS secondary server. Assume further that the DBSERVERNAME of the primary server is **myserv**, and the service name is **5656**.

To migrate all servers to the new DBSERVERNAME for the primary server:

1. Update the **sqlhosts** files on all of the secondary servers with the DBSERVERNAME and service name of the new primary server.

   Existing **sqlhosts** file:

   ```
   myserv <protocol> <host> 5656
   ```

   Modified **sqlhosts** file:

   ```
   myserv_pri <protocol> <host> 15656
   ```

2. On the Connection Manager server, edit the **sqlhosts** file as follows:

   ```
   myserv <protocol> <host> 5656
   ```

3. On the Connection Manager server, edit the **cmsm.cfg** file to set the name of the Connection Manager instance:

   ```
   NAME myserv
   SLA ...
   DEBUG ...
   LOGFILE ...
   ```

   Note that the Connection manager NAME and the corresponding **sqlhosts** file value are the same as those for the original primary server. This enables client applications to connect to Connection Manager without having to recompile applications. If you are using a version of IBM Informix Dynamic Server (IDS) earlier than version 11.50, a driver upgrade may be required. Please refer to the *IBM Informix Migration Guide*.

4. Start Connection Manager:

   ```
   oncmsm
   ```

5. Shut down the SD secondary server and the primary server.
6. On the primary server, update the **onconfig** file to configure the new DBSERVERNAME:

   Old:

   ```
   DBSERVERNAME  myserv
   ```

   New:

   ```
   DBSERVERNAME  myserv_pri
   ```

7. Start the primary server using the following command:

   ```
   oninit -SDS=myserv_pri
   ```

8. Start the SD secondary server using the following command:

   ```
   oninit
   ```

9. On the RS Secondary server, shut down the server and then start it in physical recovery mode:

   ```
   oninit -PHY
   ```

10. On the RS Secondary server, connect to new primary server with the following:

    ```
    onmode -d RSS myserv_pri
    ```

For the HDR secondary server, no additional setup is required because the HDR pair will be re-established automatically. However, steps 6 and 7 can be performed in order to register the new primary server name with the HDR secondary server.

Recovering after Failover of Primary Server in an HA Environment

This section describes the steps to recover from a situation where the primary server in a high-availability environment has failed over to a secondary server and you wish to restore the environment as it was before the failover.

Failover of the Primary Server to the HDR Secondary Server

Suppose the primary server, named **srv_pri**, has encountered an error that has caused it to fail over to an HDR secondary server named **srv_hdr_sec**. At this point, the primary server is **srv_hdr_sec**, and any other secondary servers in the cluster are now pointing to **srv_hdr_sec**. To restore the cluster to the way it was before **srv_pri** failed over, follow these steps:

1. Initialize **srv_pri** as the HDR secondary server by running the appropriate command:

   UNIX systems:

   ```
   $INFORMIXDIR/bin/hdrmksec.sh srv_hdr_sec
   ```

   Windows systems:

   ```
   hdrmksec.bat srv_hdr_sec
   ```

2. Change **srv_pri** to the primary server by running:

   ```
   onmode -d make primary srv_pri
   ```

   This command will make **srv_pri** the primary server, and will redirect any other secondary servers in the cluster to point to the new primary server. The command also shuts down the old HDR primary (**srv_hdr_sec**) because only a single primary server can exist in a high-availability environment.

3. Initialize **srv_hdr_sec** as the HDR secondary server by running the following command:

   On UNIX systems:

   ```
   $INFORMIXDIR/bin/hdrmksec.sh srv_pri
   ```

   On Windows systems:

   ```
   hdrmksec.bat srv_pri
   ```

Failover of the Primary Server to an SD Secondary Server

Now suppose that instead of the primary server failing over to an HDR secondary server, it has failed over to an SD secondary server. In this example, the primary server, named **srv_pri**, has failed over to an SD secondary server named **srv_sds_sec**. At this point, the primary server is **srv_sds_sec**, and any other

secondary servers in the cluster are now pointing to **srv_sds_sec**. To restore the cluster to the way it was before **srv_pri** failed over, follow these steps:

1. If necessary, set the following parameters in the onconfig file of **srv_pri**:

   ```
   SDS_ENABLE 1
   SDS_PAGING <path 1>,<path 2>
   SDS_TEMPDBS <dbspace_name>,<path>,<pagesize>,<offset>,<offset_size>
   ```

   See "Setting up an SD Secondary Server" on page 22-2 for more information on setting these parameters.
2. Initialize **srv_pri** as an SD secondary server by running the following command on **srv_pri**:

   ```
   oninit
   ```
3. Perform a manual failover of **srv_pri** to make it the primary server:

   ```
   onmode –d make primary srv_pri
   ```

   The above command removes **srv_sds_sec** from the cluster and makes **srv_pri** the primary server.
4. Restore **srv_sds_sec** as an SD secondary server by running the following command on **srv_sds_sec**:

   ```
   oninit
   ```

Redirection and Connectivity for Data-Replication Clients

To connect to the database servers in a replication pair, clients use the same methods with which they connect to standard database servers. For an explanation of these methods, see the descriptions of the CONNECT and DATABASE statements in the *IBM Informix Guide to SQL: Syntax*.

After a failure of one of the database servers in a pair, you might want to *redirect* the clients that use the failed database server. (You might not want clients to be redirected. For example, if you anticipate that the database servers will be functioning again in a short time, redirecting clients might not be appropriate.)

To automatically redirect clients to different database servers in a replication pair, configure applications to connect to the server group to which both HDR servers belong. When you create a connection to a server group, by default the connection is made to the current primary server in the group, if HDR is operational. If replication is down because one of the servers failed, the connection will be made to the server which is on-line (in Standard mode or Primary mode without a secondary server). You can also automate this action from within the application, as described in "Handling Redirection Within an Application" on page 19-37. Some of the client connectivity drivers included in the IBM Informix Client Software Development Kit (Client SDK) have specific mechanisms for automating redirection. For details, see the IBM Informix Client Software Development Kit (Client SDK) documentation.

Designing Clients for Redirection

When you design client applications, you must make some decisions on redirection strategies. Specifically, you must decide whether to handle redirection within the application and which redirection mechanism to use. The three different redirection mechanisms are as follows:

- Automatic redirection with the **DBPATH** environment variable
- Administrator-controlled redirection with the connectivity information

- User-controlled redirection with the **INFORMIXSERVER** environment variable

The mechanism that you employ determines which CONNECT syntax you can use in your application. The following sections describe each of the three redirection mechanisms.

Directing Clients Automatically with DBPATH

This section explains the steps that you must follow to redirect clients with the **DBPATH** environment variable and the connectivity strategy that supports this method.

What the Administrator Needs to Do

Administrators take no action to redirect clients, but they might need to attend to the type of the database server.

What the User Needs to Do

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities.

If your applications do not include such code, users who are running clients must quit and restart all applications.

How the DBPATH Redirection Method Works

When an application does not explicitly specify a database server in the CONNECT statement, and the database server that the **INFORMIXSERVER** environment variable specifies is unavailable, the client uses the **DBPATH** environment variable to locate the database (and database server).

If one of the database servers in a replication pair is unusable, applications that use that database server need not reset their **INFORMIXSERVER** environment variable if their **DBPATH** environment variable is set to the other database server in the pair. Their **INFORMIXSERVER** environment variable should always contain the name of the database server that they use regularly, and their **DBPATH** environment variable should always contain the name of the alternative database server in the pair.

For example, if applications normally use a database server called **cliff_ol**, and the database server paired with **cliff_ol** in a replication pair is called **beach_ol**, the environment variables for those applications would be as follows:

```
INFORMIXSERVER  cliff_ol
DBPATH          //beach_ol
```

Because the **DBPATH** environment variable is read only (if needed) when an application issues a CONNECT statement, applications must restart in order for redirection to occur.

An application can contain code that tests whether a connection has failed and, if so, attempts to reconnect. If an application has this code, you do not need to restart it.

You can use the CONNECT TO *database* statement with this method of redirection. For this method to work, you cannot use any of the following statements:

- CONNECT TO DEFAULT
- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

The reason for this restriction is that an application does not use **DBPATH** if a CONNECT statement specifies a database server. For more information about **DBPATH**, refer to the *IBM Informix Guide to SQL: Reference*.

Directing Clients with the Connectivity Information

This section explains the steps in redirecting clients with the connectivity information and the connectivity strategy that supports this method.

Operating System
Location of Connectivity Information

UNIX The **INFORMIXSQLHOSTS** environment variable specifies the full path name and filename of the connection information in **$INFORMIXDIR/etc/sqlhosts**. For more information about **INFORMIXSQLHOSTS**, see the *IBM Informix Guide to SQL: Reference*.

Windows
The connectivity information is in a key in the Windows registry called **HKEY_LOCAL_MACHINE\SOFTWARE \INFORMIX\SQLHOSTS**.

How Redirection with the Connectivity Information Works

The connectivity information-redirection method relies on the fact that when an application connects to a database server, it uses the connectivity information to find that database server.

If one of the database servers in a replication pair is unusable, an administrator can change the definition of the unavailable database server in the connectivity information. As described in "Changing the Connectivity Information" on page 19-34, the fields of the unavailable database server (except for the **dbservername** field) are changed to point to the remaining database server in the replication pair.

Because the connectivity information is read when a CONNECT statement is issued, applications might need to restart for redirection to occur. Applications can contain code that tests whether a connection has failed and issues a reconnect statement, if necessary. If a connection has failed, redirection is automatic, and you do not need to restart applications for redirection to occur.

Applications can use the following connectivity statements to support this method of redirection:

- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

Applications can also use the following connectivity statements, provided that the **INFORMIXSERVER** environment variable always remains set to the same database server name and the **DBPATH** environment variable is not set:

- CONNECT TO DEFAULT
- CONNECT TO *database*

Changing the Connectivity Information

To use the connectivity information to redirect clients, you must change the connectivity information for the clients and change other connectivity files, if necessary.

For more information, refer to "Configuring HDR Connectivity" on page 20-5 and Chapter 3, "Client/Server Communications," on page 3-1.

To change the connectivity information on the client computer

1. Comment out the entry for the failed database server.
2. Add an entry that specifies the dbservername of the failed database server in the **servername** field and information for the database server to which you are redirecting clients in the **nettype**, **hostname**, and **servicename** fields.
3. Use the following options in the **sqlhosts** file or registry to redirect applications to another database server if a failure should occur:
   a. "Connection-Redirection Option" on page 3-23
   b. "End-of-Group Option" on page 3-24
   c. "Group Option" on page 3-24

Figure 19-27 on page 19-35 shows how connectivity values might be modified to redirect clients.

You do not need to change entries in the connectivity information on either of the computers that is running the database servers.

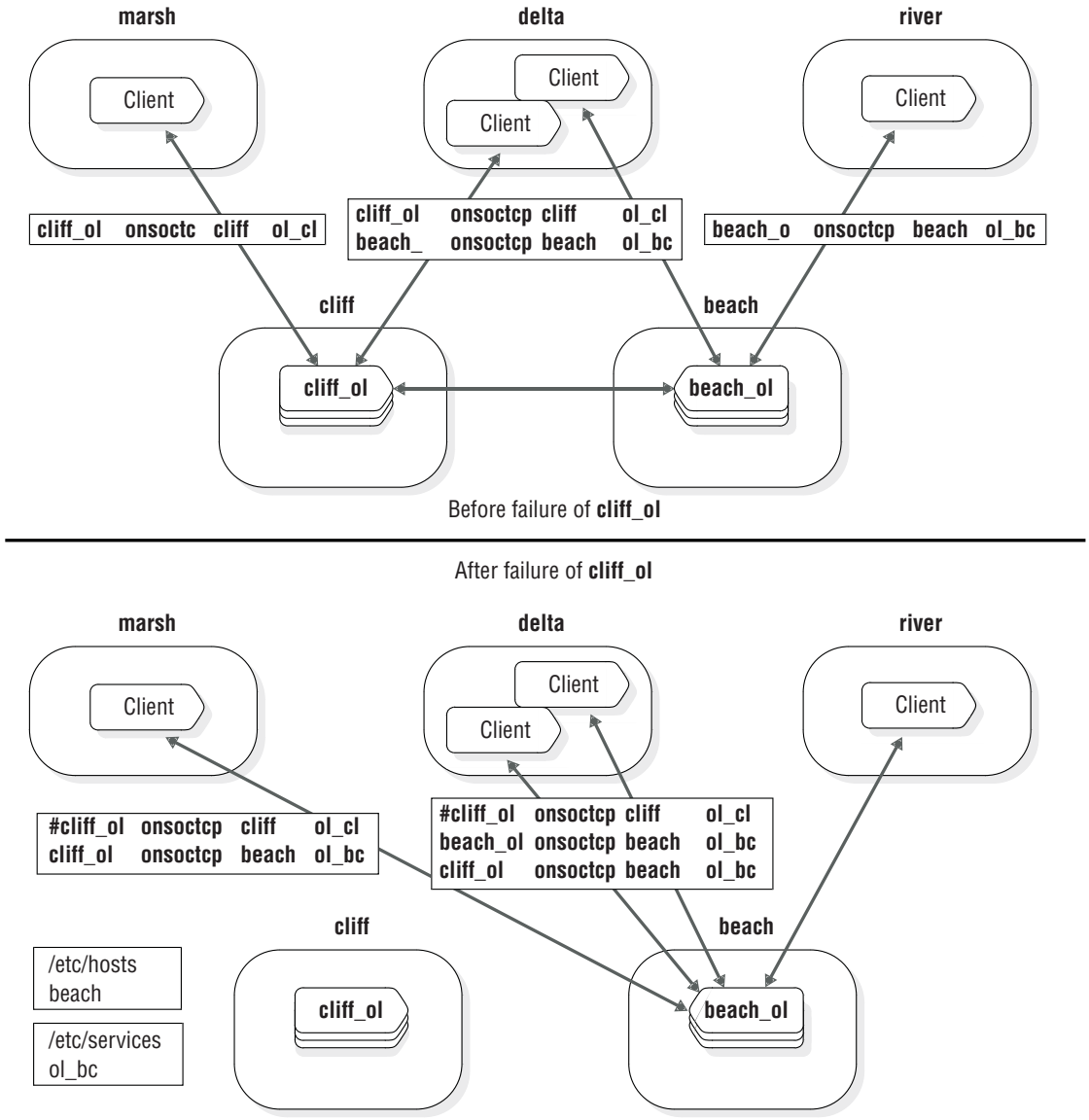


Figure 19-27. Connectivity Values Before and After a Failure of the cliff_ol Database Server

To change other connectivity files

You also must ensure that the following statements are true on the client computer before that client can reconnect to the other database server.

1. The **/etc/hosts** file on UNIX or **hosts** file on Windows has an entry for the **hostname** of the computer that is running the database server to which you are redirecting clients.
2. The **/etc/services** file on UNIX or **services** file on Windows has an entry for the **servicename** of the database server to which you are redirecting clients.

Connecting to the Database Server

After the administrator changes the connectivity information and other connectivity files (if needed), clients connect to the database server to which the administrator redirects them when they issue their next CONNECT statement.

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities. If your applications do not include such code, users who are running clients must quit and restart all applications.

Automatic Redirection with Server Groups

You can use the group option in the SQLHOSTS file to specify a server group to which applications connect instead of an individual database server. To make connection redirection automatic, add a database server definition for both the primary and secondary servers to the server group definition. By default, when a connection request is made to an HDR server group, the connection is routed to the primary server. If the primary server is unavailable, then the connection request will be routed to the secondary server that gets promoted to the primary server after failover processing.

For example, the following SQLHOSTS entries represent an HDR server group, **g_hdr**, with a primary server definition, **hdr_prim**, and a secondary server definition, **hdr_sec**.

Table 19-4. SQLHOSTS entries for an HDR Server Group

dbservername	nettype	hostname	servicename	options
g_hdr	group	-	-	i=1
hdr_prim	ontlitcp	machine1pri	port1	g=g_hdr
hdr_sec	ontlitcp	machine1sec	port1	g=g_hdr

Applications can use the following connectivity statements to support this method of redirection:

- CONNECT TO *database@dbserver_group*
- CONNECT TO *@dbserver_group*

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities. If your applications do not include such code, users who are running clients must quit and restart all applications.

Directing Clients with INFORMIXSERVER

This section explains the steps in redirecting clients with the **INFORMIXSERVER** environment variable and the connectivity strategy that supports this method.

How Redirection Works with INFORMIXSERVER

The **INFORMIXSERVER** redirection method relies on the fact that when an application does not explicitly specify a database server in the CONNECT statement, the database server connects to the client that the **INFORMIXSERVER** environment variable specifies.

If one of the database servers in a replication pair is unusable, applications that use that database server can reset their **INFORMIXSERVER** environment variable to the other database server in the pair to access the same data.

Applications read the value of the **INFORMIXSERVER** environment variable only when they start. Therefore, applications must be restarted to recognize a change in the environment variable.

To support this method of redirection, you can use the following connectivity statements:

- CONNECT TO DEFAULT
- CONNECT TO *database*

You cannot use the CONNECT TO *database@dbserver* or CONNECT TO *@dbserver* statements for this method. When a database server is explicitly named, the CONNECT statement does not use the **INFORMIXSERVER** environment variable to find a database server.

What the Administrator Needs to Do

Administrators take no action to redirect the clients, but they might need to change the type of the database server.

What the User Needs to Do

Users who are running client applications must perform the following three steps when they decide to redirect clients with the **INFORMIXSERVER** environment variable.

To redirect clients with the INFORMIXSERVER environment variable

1. Quit their applications.
2. Change their **INFORMIXSERVER** environment variable to hold the name of the other database server in the replication pair.
3. Restart their applications.

Handling Redirection Within an Application

If you use **DBPATH** or connectivity information to redirect, you can include in your clients a routine that handles errors when clients encounter an HDR failure. The routine can call another function that contains a loop that tries repeatedly to connect to the other database server in the pair. This routine redirects clients without requiring the user to exit the application and restart it.

Figure 19-28 on page 19-38 shows an example of a function in a client application using the **DBPATH** redirection mechanism that loops as it attempts to reconnect. After it establishes a connection, it also tests the type of the database server to make sure it is not a secondary database server. If the database server is still a secondary type, it calls another function to alert the user (or database server administrator) that the database server cannot accept updates.

```
/* The routine assumes that the INFORMIXSERVER environment
 * variable is set to the database server that the client
 * normally uses and that the DBPATH environment variable
 * is set to the other database server in the pair.
 */

#define SLEEPTIME 15
#define MAXTRIES 10

main()
{
   int connected = 0;
   int tries;
   for (tries = 0;tries < MAXTRIES && connected == 0;tries++)
   {
      EXEC SQL CONNECT TO "superstores";
      if (strcmp(SQLSTATE,"00000"))
      {
         if (sqlca.sqlwarn.sqlwarn6 != 'W')
         {
            notify_admin();
            if (tries < MAXTRIES - 1)
               sleep(SLEEPTIME);
            }
         else connected =1;
         }
      }
   return ((tries == MAXTRIES)? -1:0);
   }
```

Figure 19-28. Example of a CONNECT Loop for DBPATH Redirection Mechanism

This example assumes the **DBPATH** redirection mechanism and uses a form of the CONNECT statement that supports the **DBPATH** redirection method. If you used the connectivity information to redirect, you might have a different connection statement, as follows:

```
EXEC SQL CONNECT TO "superstores@cliff_ol";
```

In this example, **superstores@cliff_ol** refers to a database on a database server that the client computer recognizes. For redirection to occur, the administrator must change the connectivity information to make that name refer to a different database server. You might need to adjust the amount of time that the client waits before it tries to connect or the number of tries that the function makes. Provide enough time for an administrative action on the database server (to change the connectivity information or change the type of the secondary database server to standard).

Comparison of Different Redirection Mechanisms

Table 19-5 summarizes the differences among redirection mechanisms.

Table 19-5. Comparison of Redirection Methods for Different Connectivity Strategies

	Automatic Redirection	**User Redirection**
DBPATH		
When is a client redirected?	When the client next tries to connect with a specified database	When the client next tries to connect with a specified database
Do clients need to be restarted to be redirected?	No	Yes
What is the scope of the redirection?	Individual clients redirected	Individual clients redirected

Table 19-5. Comparison of Redirection Methods for Different Connectivity Strategies (continued)

	Automatic Redirection	**User Redirection**
Are changes to environment variables required?	No	No
Connectivity Information		
When is a client redirected?	After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server	After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server
Do clients need to be restarted to be redirected?	No	Yes
What is the scope of the redirection?	All clients that use a given database server redirected	Individual clients redirected
Are changes to environment variables required?	No	No
INFORMIXDIR		
When is a client redirected?		When the client restarts and reads a new value for the **INFORMIXSERVER** environment variable
Do clients need to be restarted to be redirected?		Yes
What is the scope of the redirection?		Individual clients redirected
Are changes to environment variables required?		Yes
Connection Manager		
When is a client redirected?		When the configured service level agreement is attained.
Do clients need to be restarted to be redirected?		No
What is the scope of the redirection?		Individual clients redirected
Are changes to environment variables required?		No

Designing Data Replication Group Clients

This section discusses various design considerations for clients that connect to database servers that are running data replication.

Also see "Isolation Levels on Secondary Servers" on page 19-13 for information about **committed read** and **committed read last committed** isolation levels on secondary servers.

Use of Temporary Dbspaces for Sorting and Temporary Tables

Even though the secondary database server is in read-only mode, it does write when it needs to sort or create a temporary table. "Temporary Dbspaces" on page 9-10 explains where the database server finds temporary space to use during a sort or for a temporary table. To prevent the secondary database server from writing to a dbspace that is in logical-recovery mode, you must take the following actions:

1. Ensure that one or more temporary dbspaces exist. For instructions on creating a temporary dbspace, see "Creating a Dbspace that Uses the Default Page Size" on page 10-7.
2. Perform one of the following actions:
   - Set the DBSPACETEMP parameter in the ONCONFIG file of the secondary database server to the temporary dbspace or dbspaces.
   - Set the **DBSPACETEMP** environment variable of the client applications to the temporary dbspace or dbspaces.

Temporary tables created on secondary servers (SD secondary servers, RS secondary servers, and HDR secondary servers) should be created using the WITH NO LOG option. Alternatively, set the TEMPTAB_NOLOG configuration parameter to 1 on the SD secondary server to change the default logging mode for temporary tables to no logging. Tables created with logging enabled will result in ISAM errors.

If you are using an SD secondary server, set the TEMPTAB_NOLOG configuration parameter 1 on the SD secondary server to change the default logging mode for temporary tables to no logging. Otherwise user may get ″140: ISAM error: operation illegal on a DR Secondary″ error unless temporary table is create with ″no log″ clause.

For SD secondary servers, set the SDS_TEMPDBS configuration parameter for configuring temporary dbspaces to be used by the SD secondary server.

For SD secondary servers, it is not necessary to explicitly add a temporary dbspace because the secondary server will allocate the chunk specified by SDS_TEMPDBS when the server is started. It is only necessary to prepare the device that will accept the chunk.

On SD secondary servers, you can set the DBSPACETEMP configuration parameter to the same value as DBSPACETEMP on the primary server. In the event that the SD secondary server fails over to the primary server, applications will use the name specified in DBSPACETEMP instead of SDS_TEMPDBS.

Chapter 20. Using High-Availability Data Replication (Enterprise/Workgroup Editions)

In This Chapter

This chapter describes how to plan, configure, start, and monitor High-Availability Data Replication (HDR) for Dynamic Server, and how to restore data after a media failure. If you plan to use HDR, read this entire chapter first. If you plan to use IBM Informix Enterprise Replication, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Chapter 19, "Data Replication Overview (Enterprise/Workgroup Editions)," on page 19-1, explains what HDR is, how it works, and how to design client applications for an HDR environment.

HDR is available with the standard version of Dynamic Server. HDR does not work with the IBM Informix Dynamic Server Express Edition.

Planning for HDR

Before you start setting up computers and database servers to use HDR, you might want to do some initial planning. The following list contains planning tasks to perform:

- Choose and acquire appropriate hardware.
- If you are using more than one database server to store data that you want to replicate, migrate and redistribute this data so that it can be managed by a single database server.
- Ensure that all databases you want to replicate use transaction logging. To turn on transaction logging, see Chapter 12, "Managing the Database-Logging Mode," on page 12-1.
- Develop client applications to make use of both database servers in the replication pair. For a discussion of design considerations, refer to "Redirection and Connectivity for Data-Replication Clients" on page 19-31 and "Designing Data Replication Group Clients" on page 19-39.
- Create a schedule for starting HDR for the first time.
- Design a storage-space and logical-log backup schedule for the primary database server.
- Produce a plan for how to handle failures of either database server and how to restart HDR after a failure. Read "Redirection and Connectivity for Data-Replication Clients" on page 19-31.

Configuring a System for HDR

To configure your system for HDR, you must take the following actions:

- Meet hardware and operating-system requirements.
- Meet database and data requirements.
- Meet database server configuration requirements.
- Configure HDR connectivity.

Each of these topics is explained in this section.

You can configure your system to use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures the privacy and integrity of data transmitted over the network, for HDR communications. You can use the SSL protocol for connections between primary and secondary servers and for connections with remote standalone (RS) and shared disk (SD) secondary servers in a high-availability configuration. For information on using the SSL protocol, see the "Secure Sockets Layer Communication Protocol Encryption" section of the *IBM Informix Security Guide*.

The HDR Connection Manager also supports Distributed Relational Database Architecture (DRDA) connections. For more information, see "Distributed Relational Database Architecture (DRDA) Communications" on page 3-36.

Hardware and Operating-System Requirements for HDR

For an HDR database server pair to function, your hardware must meet certain requirements.

Your hardware must meet the following requirements:

- The primary and secondary servers must be able to run the same Dynamic Server executable image, even if they do not have identical hardware or operating systems. For example, you can use servers with different Linux 32-bit operating systems because those operating systems can run the same Dynamic Server executable image. In this situation, you cannot add a server on a Linux 64-bit operating system because that operating system requires a different Dynamic Server executable image. Check the machine notes file: you can use any combination of hardware and operating systems listed as supported in the same machine notes file.
- The hardware that runs the primary and secondary database servers must support network capabilities.
- The amount of disk space allocated to dbspaces for the primary and secondary database servers must be equal. The type of disk space is irrelevant; you can use any mixture of raw or cooked spaces on the two database servers.
- The chunks on each computer must have the same path names. Symbolic links are allowed for UNIX platforms, but not for Windows platforms.

Database and Data Requirements for HDR

For an HDR database server pair to function, your database and data must meet certain requirements.

Your database and data must meet the following requirements:

- **All data must be logged.**

  All databases that you want to replicate must have transaction logging turned on.

  This requirement is important because the secondary database server uses logical-log records from the primary database server to update the data that it manages. If databases managed by the primary database server do not use logging, updates to those databases do not generate log records, so the secondary database server has no means of updating the replicated data. Logging can be buffered or unbuffered.

  If you need to turn on transaction logging before you start HDR, see "Turning On Transaction Logging with ontape" on page 12-3.
- **The data must reside in dbspaces or sbspaces.**

If your primary database server has simple large objects stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal HDR processing. However, simple-large-object data within dbspaces is replicated.

Smart large objects, which are stored in sbspaces, are replicated. The sbspaces must be logged. User-defined types (UDTs) are replicated, unless they have out-of-row data stored in operating system files. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server.

Requirements for Compression Operations When You Use HDR

If you use the Dynamic Server disk compression feature, data that is compressed in the source table will be compressed in the target table. You cannot perform compression operations on an HDR secondary, RS secondary, or SD secondary server, because the HDR target server must have exactly the same data and physical layout as the source server.

Meeting Database Server Configuration Requirements

For an HDR database server pair to function, you must fully configure each of the database servers. For information on configuring a database server, refer to Chapter 1, "Installing and Configuring the Database Server," on page 1-1. You can then use the relevant aspects of that configuration to configure the other database server in the pair. For more information on the configuration parameters, see the *IBM Informix Administrator's Reference*.

This section describes the following configuration considerations for HDR database server pairs:

- "Database Server Version"
- "Storage Space and Chunk Configuration"
- "Mirroring" on page 20-4
- "Physical-Log Configuration" on page 20-4
- "Dbspace and Logical-Log Tape Backup Devices" on page 20-4
- "Logical-Log Configuration" on page 20-5
- "HDR Configuration Parameters" on page 20-5

Database Server Version

The versions of the database server on the primary and secondary database servers must be identical.

Storage Space and Chunk Configuration

The number of dbspaces, the number of chunks, their sizes, their path names, and their offsets must be identical on the primary and secondary database servers. In addition, the configuration must contain at least one temporary dbspace if the HDR secondary server is used for creating activity reports. See "Use of Temporary Dbspaces for Sorting and Temporary Tables" on page 19-39.

UNIX Only:

You should use symbolic links for the chunk path names, as explained in "Allocating Raw Disk Space on UNIX" on page 10-3.

Important: If you do not use symbolic links for chunk path names, you cannot easily change the path name of a chunk. For more information, see "Renaming Chunks" on page 20-11.

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME
- ROOTOFFSET
- ROOTPATH
- ROOTSIZE

Using Non-default Page Sizes in an HDR Environment

The page size of a dbspace and the buffer pool specifications are automatically propagated from the primary to the secondary database server. While both the primary and the secondary database servers must have the same buffer pools, the number of buffers in the buffer pools do not need to match.

Mirroring

You do not have to set the MIRROR parameter to the same value on the two database servers; you can enable mirroring on one database server and disable mirroring on the other. However, if you specify a mirror chunk for the root chunk of the primary database server, you must also specify a mirror chunk for the root chunk on the secondary database server. Therefore, the following ONCONFIG parameters must be set to the same value on both database servers:

- MIRROROFFSET
- MIRRORPATH

Physical-Log Configuration

The physical log should be identical on both database servers. The following ONCONFIG parameters must have the same value on each database server:

- PHYSBUFF
- PHYSFILE

Dbspace and Logical-Log Tape Backup Devices

You can specify different tape devices for the primary and secondary database servers.

If you use ON–Bar, set the ON–Bar configuration parameters to the same value on both database servers. For information on the ON–Bar parameters, see the *IBM Informix Backup and Restore Guide*.

If you use **ontape**, the tape size and tape block size for the storage-space and logical-log backup devices should be identical. The following ONCONFIG parameters must have the same value on each database server:

- LTAPEBLK
- LTAPESIZE
- TAPEBLK
- TAPESIZE

To use a tape to its full physical capacity, set LTAPESIZE and TAPESIZE to `0`.

Logical-Log Configuration

All log records are replicated to the secondary server. You must configure the same number of logical-log files and the same logical-log size for both database servers. The following ONCONFIG parameters must have the same value on each database server:

- LOGBUFF
- LOGFILES
- LOGSIZE
- DYNAMIC_LOGS

The database server logs the addition of logical-log files. Logical-log files added dynamically on the primary server are automatically replicated on the secondary server. Although the DYNAMIC_LOGS value on the secondary server has no effect, keep DYNAMIC_LOGS in sync with the value on the primary server, in case their roles switch.

HDR Configuration Parameters

The following HDR configuration parameters must be set to the same value on both database servers in the replication pair:

- DRAUTO
- DRINTERVAL
- DRTIMEOUT

Configuring HDR Connectivity

For an HDR database server pair to function, the two database servers must be able to establish a connection with one another. To satisfy this requirement, the connectivity information on each of the computers that is running the database server in a replication pair must have at least the following entries:

- An entry that identifies the database server that is running on that computer
- An entry that identifies the other database server in the data-replication pair

Important: Set the **nettype** field of the **sqlhosts** file or registry and the NETTYPE configuration parameter to a network protocol such as **ontlitcp**, **onsoctcp**, or **ontlispx** so that the database servers on two different computers can communicate with each other. HDR does not work if the **nettype** field specifies a non-network protocol such as **onipcshm**, **onipcstr**, or **onipcnmp**.

For information on how to redirect clients and change connectivity information, see "Redirection and Connectivity for Data-Replication Clients" on page 19-31.

Starting HDR for the First Time

After you complete the HDR configuration, you are ready to start HDR. This section describes the necessary steps for starting HDR.

Suppose you want to start HDR on two database servers, **ServerA** and **ServerB**. The procedure for starting HDR, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps. Table 20-1 on page 20-7 lists the commands required to perform each step and the

messages sent to the message log. You can use **ontape** or ON–Bar to perform the backup and restore. You must use the same utility throughout the procedure.

Important: If you use ON–Bar to perform the backup and restore, **ontape** is required on both database servers. You cannot remove **ontape** from database servers participating in HDR.
If desired, you can also set up HDR using external backup and restore. See the *IBM Informix Backup and Restore Guide* for information on how to perform an external backup and restore. See "Decreasing Setup Time Using the ontape STDIO Feature" on page 20-8 for the quickest way to set up your HDR secondary directly from the HDR primary.

To start HDR

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.
2. Create a level-0 backup of **ServerA**.
3. Use the **onmode -d** command to set the type of **ServerA** to primary and to indicate the name of the associated secondary database server (in this case **ServerB**).

   When you issue an **onmode -d** command, the database server attempts to establish a connection with the other database server in the HDR pair and to start HDR operation. The attempt to establish a connection succeeds only if the other database server in the pair is already set to the correct type.

   At this point, **ServerB** is not online and is not set to type secondary, so the HDR connection is not established.
4. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 1. Do not perform a logical restore.

   If you are using:
   - ON–Bar, use the **onbar -r -p** command to perform a physical restore.
   - ON–Bar and performing an external restore, use the **onbar -r -p -e** command to perform the physical restore.
   - **ontape**, use the **ontape -p** option. You cannot use the **ontape -r** option because it performs both a physical and a logical restore.
   - **ontape** and performing an external restore, use the **ontape -p -e** command to perform the physical restore.
5. Use the **onmode -d** command to set the type of **ServerB** to secondary and indicate the associated primary database server.

   ServerB tries to establish an HDR connection with the primary database server (**ServerA**) and start operation. The connection should be successfully established.

   Before HDR begins, the secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 2. If all these logical-log records still reside on the primary database server disk, the primary database server sends these records directly to the secondary database server over the network and logical recovery occurs automatically.

   If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 6 on page 20-7.

Important: You must complete steps 4 on page 20-6 and 5 on page 20-6 during the same session. If you need to shut down and restart the secondary database server after step 4 on page 20-6, you must redo step 4 on page 20-6.

6. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.

   If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

   After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

Table 20-1 illustrates the preceding steps so that you can clearly determine which steps are performed on the primary server and which are performed on the secondary server. The table also shows information written to the log file after each step is performed.

Table 20-1. Steps to Start HDR for the First Time

Step	On the Primary	On the Secondary
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**ontape command** ontape -s -L 0 **ON–Bar command** onbar -b -L 0 **Messages to message log** Level 0 archive started on rootdbs. Archive on rootdbs completed.	
3	**onmode command** onmode -d primary *sec_name* **Messages to message log** DR: new type = primary server name = *sec_name* DR: Trying to connect to secondary server DR: Cannot connect to secondary server	

Table 20-1. Steps to Start HDR for the First Time (continued)

Step	On the Primary	On the Secondary
4.		**ontape command** ontape -p or ontape -p -e Answer no when you are prompted to back up the logs. **ON–Bar command** onbar -r -p or onbar -r -p -e **Messages to message log** IBM Informix Database Server Initialized -- Shared Memory Initialized Recovery Mode Physical restore of rootdbs started. Physical restore of rootdbs completed.
5.		**onmode command** onmode -d secondary *prim_name* **Messages to message log** DR: new type = secondary server name = *prim_name* If all the logical-log records written to the primary database server since step 1 still reside on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 6 must be performed).
	Messages to message log DR: Primary server connected DR: Primary server operational	**Messages to message log** DR: Trying to connect to primary server DR: Secondary server connected DR: Failure recovery from disk in process. *n* recovery worker threads will be started. Logical Recovery Started Start Logical Recovery - Start Log *n*, End Log? Starting Log Position - *n 0xnnnnn*DR: Secondary server operational
6.		**ontape command** ontape -l **ON–Bar command** onbar -r -l
	Messages to message log DR: Primary server connected DR: Primary server operational	**Messages to message log** DR: Secondary server connected DR: Failure recovery from disk in process. *n* recovery worker threads will be started. Logical Recovery Started Start Logical Recovery - Start Log *n*, End Log? *Starting Log Position - n 0xnnnnn*DR: Secondary server operational

Decreasing Setup Time Using the ontape STDIO Feature

You can dramatically improve the speed of setting up HDR by using the **ontape** STDIO feature. Using this feature **ontape** will write the data to the shell's standard output during a backup, and then read it from standard input during a restore. Combining a STDIO backup with a simultaneous STDIO restore in a pipe using a

remote command interpreter (such as rsh or ssh), allows performing the initial setup of an HDR (or RSS) secondary server using a single command line. This saves storage space by not having to write to or read from tape or disk, and does not require waiting for the backup to finish before the restore can start.

See the *IBM Informix Backup and Restore Guide* for details about using the `STDIO` value.

This method for setting up HDR using **ontape** can be used regardless of which backup utility is used (**ontape** or **ON–Bar**).

Important: When you use STDIO in this way, no persistent backup is saved anywhere that could be used to perform a restore. The use of the -F (fake) option on the source (backup) side will not record the backup in the database server's reserved pages. Also, any interactive dialog is suppressed and no prompts or questions will appear. You must also ensure that the remote part of the pipe picks the proper environment for the remote IDS instance. The script must not produce any output other than the backup data since this would be read by the restore process (for example, do not enable tracing).

The steps in the following table must be performed by user **informix**, the scripts should be executable, and, if called without a complete path, should reside in your home directory. You can use ssh instead of rsh if you require secure data transmission across the network.

Table 20-2. Alternate Method of Setting Up HDR from the Primary Server, using rsh

Step	On the Primary	On the Secondary
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**onmode command** onmode -d primary *sec_name*	
3	**ontape command** ontape -s -L 0 -t STDIO -F | rsh <secondary_machine> ontape_HDR_restore.ksh	
4.		**onmode command** onmode -d secondary *pri_name*

In the above table, the script **ontape_HDR_restore.ksh** on the secondary server should contain the following commands:

```
#!/bin/ksh
# first get the proper IDS environment set
. hdr_sec.env
# redirecting stdout and stderr required since otherwise command might never return
ontape -p -t STDIO > /dev/null 2>&1
```

The following steps show how to set up HDR from the secondary server.

Table 20-3. Alternate Method of Setting Up HDR from the Secondary Server, using rsh:

Step	On the Primary	On the Secondary
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**onmode command** onmode -d primary *sec_name*	
3		**ontape command** rsh <primary_machine> ontape_HDR_backup.ksh | ontape -p -t STDIO
4.		**onmode command** onmode -d secondary *pri_name*

In the above table, the script **ontape_HDR_backup.ksh** on the primary server should contain the following commands:

```
#!/bin/ksh
# first get the proper IDS environment set
. hdr_pri.env
ontape -s -L 0 -F -t STDIO
```

Performing Basic Administration Tasks

This section contains instructions on how to perform database server administration tasks when your system is running HDR.

Changing Database Server Configuration Parameters

Some of the configuration parameters must be set to the same value on both database servers in the replication pair (as listed under "Meeting Database Server Configuration Requirements" on page 20-3.) Other Dynamic Server configuration parameters can be set to different values.

To make changes to ONCONFIG files

1. Bring each database server offline with the **onmode -k** option. If DRAUTO is set to RETAIN_TYPE or REVERSE_TYPE, you can more easily bring the secondary database server off-line first.
2. Change the parameters on each database server.
3. Starting with the last database server that you brought offline, bring each database server back online.

   For example, if you brought the secondary database server offline last, bring the secondary database server online first. Table 20-1 on page 20-7 lists the procedures for bringing the primary and secondary database servers back online.

If the configuration parameter does not need to have the same value on each database server in the replication pair, you can change the value on the primary or secondary database server individually.

Backing Up Storage Spaces and Logical-Log Files

When you use HDR, you must back up logical-log files and storage spaces only on the primary database server. Be prepared, however, to perform storage-space and logical-log backups on the secondary database server in case the type of the database server is changed to standard.

You must use the same backup and restore tool on both database servers.

The block size and tape size used (for both storage-space backups and logical-log backups) must be identical on the primary and secondary database servers.

You can use **ontape** to set the tape size to `0` to automatically use the full physical capacity of a tape.

Changing the Logging Mode of Databases

You cannot turn on transaction logging for databases on the primary database server while you are using HDR. You can turn logging off for a database; however, subsequent changes to that database are not duplicated on the secondary database server.

To turn on database logging

1. To turn HDR off, shut down the secondary database server.
2. Turn on database logging.

   After you turn on logging for a database, if you start data replication without performing the level-0 backup on the primary database server and restore on the secondary database server, the database on the primary and secondary database servers might have different data. This situation could cause data-replication problems.
3. Perform a level-0 backup on the primary database server and restore on the secondary database server. This procedure is described in "Starting HDR for the First Time" on page 20-5.

Adding and Dropping Chunks and Storage Spaces

You can perform disk-layout operations, such as adding or dropping chunks and dbspaces, only from the primary database server. The operation is replicated on the secondary database server. This arrangement ensures that the disk layout on both database servers in the replication pair remains consistent.

The directory path name or the actual file for chunks must exist before you create them. Make sure the path names (and offsets, if applicable) exist on the secondary database server before you create a chunk on the primary database server, or else the database server turns off data replication.

Renaming Chunks

If you use symbolic links for chunk path names, you can rename chunks while HDR is operating. For instructions on renaming chunks, see the *IBM Informix Backup and Restore Guide*.

If you do not use symbolic links for chunk path names, you must take both database servers offline while renaming the chunks, for the time that it take to complete a cold restore of the database server.

To rename chunks on a failed HDR server

1. Change the mode of the undamaged server to standard.
2. Take a level-0 backup of the standard server.
3. Shut down the standard server.
4. Rename the chunks on the standard server during a cold restore from the new level-0 archive (for instructions, see the *IBM Informix Backup and Restore Guide*).
5. Start the standard server.
6. Take another level-0 archive of the standard server. Be sure the server is in standard mode.
7. Restore the failed server with the new level-0 backup and reestablish the HDR pair.

Saving Chunk Status on the Secondary Database Server

For a data-replication pair, if the status of a chunk (*down*, *online*) is changed on the secondary database server, and that secondary server is restarted before a checkpoint is completed, the updated chunk status is not saved.

To ensure that the new chunk status is flushed to the reserved pages on the secondary database server, force a checkpoint on the primary database server and verify that a checkpoint also completes on the secondary database server. The new chunk status is now retained even if the secondary database server is restarted.

If the primary chunk on the secondary database server is down, you can recover the primary chunk from the mirror chunk.

To recover the primary chunk from the mirror chunk

1. Run **onspaces -s** on the secondary database server to bring the primary chunk online.

   You also can use ISA to bring the primary chunk online.
2. Run **onmode -c** on the primary database server to force a checkpoint.
3. Run **onmode -m** on the primary database server to verify that a checkpoint was actually performed.
4. Run **onmode -m** on the secondary database server to verify that a checkpoint was also completed on the secondary database server.

After you complete these steps, the primary chunk will be online when you restart the secondary database server.

Using and Changing Mirroring of Chunks

Before you can add a mirror chunk, the disk space for that chunk must already be allocated on both the primary and secondary database servers. If you want to mirror a dbspace on one of the database servers in the replication pair, you must create mirror chunks for that dbspace on *both* database servers. For general information on allocating disk space, see "Allocating Disk Space" on page 10-1.

Do not set the MIRROR configuration parameter to 1 unless you are using mirroring.

You can perform disk-layout operations from the primary database server only. Thus, you can add or drop a mirror chunk only from the primary database server. A mirror chunk that you add to or drop from the primary database server is added to or dropped from the secondary database server as well. You must perform mirror recovery for the newly added mirror chunk on the secondary database server. For more information, see "Recovering a Mirror Chunk" on page 18-6.

When you drop a chunk from the primary database server, Dynamic Server automatically drops the corresponding chunk on the secondary database server. This applies to both primary and mirror chunks.

When you turn mirroring off for a dbspace on the primary database server, Dynamic Server does not turn mirroring off for the corresponding dbspace on the secondary database server. To turn off mirroring for a dbspace on the secondary database server independent of the primary server, use **onspaces -r**. For more information about turning off mirroring, see "Ending Mirroring" on page 18-6.

You can take down a mirror chunk or recover a mirror chunk on either the primary or secondary database server. These processes are transparent to HDR.

Managing the Physical Log

The size of the physical log must be the same on both database servers. If you change the size and location of the physical log on the primary database server, this change is replicated to the secondary database server. ONCONFIG values on secondary are updated automatically.

For information on changing the size and location of the physical log, refer to Chapter 16, "Managing the Physical Log," on page 16-1.

Managing the Logical Log

The size of the logical log must be the same on both database servers. You can add or drop a logical-log file with the **onparams** utility, as described in Chapter 14, "Managing Logical-Log Files," on page 14-1. Dynamic Server replicates this change on the secondary database server; however, the LOGFILES parameter on the secondary database server is not updated. After you issue the **onparams** command from the primary database server, therefore, you must manually change the LOGFILES parameter to the desired value on the secondary database server. Finally, for the change to take effect, you must perform a level-0 backup of the root dbspace on the primary database server.

If you add a logical-log file to the primary database server, this file is available for use and flagged `F` as soon as you perform the level-0 backup. The new logical-log file on the secondary database server is still flagged `A`. However, this condition does not prevent the secondary database server from writing to the file.

Managing Virtual Processors

The number of virtual processors has no effect on data replication. You can configure and tune each database server in the pair individually.

Managing Shared Memory

If you make changes to the shared-memory ONCONFIG parameters on one database server, you must make the same changes at the same time to the shared-memory ONCONFIG parameters on the other database server. For the procedure for making this change, see "Changing Database Server Configuration Parameters" on page 20-10.

Replicating an Index to an HDR Secondary Database Server

If index page logging is enabled, index replication to the HDR secondary database server occurs automatically (see "Index Page Logging" on page 21-3). If index page logging is disabled, and an index on an HDR secondary database server becomes corrupt and needs to be rebuilt, you can either:

- Manually replicate the index from the primary server to the secondary server.
- Let the secondary server automatically replicate the index if you enabled the secondary server to do this.

To enable the secondary database server to automatically replicate the index, either:

- Set **onmode -d idxauto** to `on`.
- Set the value of the DRIDXAUTO configuration parameter to `1`.

After you set either of these values, when one of the threads on the secondary database server detects a corrupt index, the index is automatically replicated to the secondary database server. Restarting index replication can take up to the amount of time specified in seconds in the DRTIMEOUT configuration parameter.

Sometimes, you might want to replicate an index manually, for example, when you want to postpone index repair because the table is locked. If you want to be able to manually replicate an index on the HDR secondary server, turn off the automatic replication feature.

To turn off the automatic index replication feature, either:

- Set **onmode -d idxauto** to `off`.
- Set the DRIDXAUTO configuration parameter to `0`.

If **onmode -d idxauto** is set to off or DRIDXAUTO is set to `0` and the secondary server detects a corrupt index, you can manually replicate an index on the HDR secondary server by issuing an **onmode -d index** command in the following format.

```
onmode -d index database:[ownername].table#index
```

For example:

```
onmode -d index cash_db:user_dx.table_12#index_z
```

In the case of a fragmented index with one corrupt fragment, the **onmode -d idxauto** option only transfers the single affected fragment, whereas the **onmode -d index** option transfers the whole index.

Note: When turning the automatic index replication feature on or off, you can use either the **onmode** command or the DRIDXAUTO configuration parameter. If you use the **onmode** command, you do not need to stop and restart the database

server. When you use the DRIDXAUTO parameter, the database server is restarted with the setting you specify. The **onmode** command does not change the DRIDXAUTO value. If you use the **onmode** command, you must manually change the value of DRIDXAUTO.

The **online.log** file produced by the secondary server contains information about any replicated index.

Encrypting Data Traffic Between HDR Database Servers

You can use Dynamic Server encryption options to encrypt the data traffic between the database servers of an HDR pair. Do this when you want to ensure secure transmission of data.

After you enable encryption, the first database server in an HDR pair encrypts the data before sending the data to the other server in the pair. The server that receives the data, decrypts the data as soon as it receives the data.

For updatable secondary servers in a high-availability cluster environment, encryption from the updatable secondary server to primary server requires SMX encryption. To encrypt data sent from an updatable secondary server to the primary server, set the **ENCRYPT_SMX** configuration parameter on the secondary server. See "Enabling SMX Encryption" on page 21-4 for more information.

Prerequisites: To support encrypted HDR connections in conjunction with Communication Support Module (CSM) client/server encryption, two network ports must be configured:

- One network port must be configured for HDR.
- The other network port must be configured for CSM client/server connections.

Important: You cannot start HDR on a network connection that is configured to use CSM encryption for client/server connections.

Additional buffers or larger buffers might be necessary to accommodate the size of encrypted data.

To encrypt data traffic between two HDR database servers:

1. Set the following configuration parameters on the first server in the HDR pair.
   - ENCRYPT_HDR, which enables or disables HDR encryption
   - ENCRYPT_CIPHERS, which specifies the ciphers and modes to use for encryption
   - ENCRYPT_MAC, which controls the level of message authentication code (MAC) generation
   - ENCRYPT_MACFILE, which specifies a list of the full path names of MAC key files
   - ENCRYPT_SWITCH, which specifies the number of minutes between automatic renegotiations of ciphers and keys

   To change these parameters, follow the instructions in "Changing Database Server Configuration Parameters" on page 20-10.
2. Set the encryption configuration parameters on the secondary server. The ENCRYPT_HDR, ENCRYPT_CIPHERS, ENCRYPT_MAC, and the ENCRYPT_SWITCH configuration parameters must have the same values as the corresponding configuration parameters on the primary server. The

ENCRYPT_MACFILE configuration parameter can have a different value on each server, but the files must contain the same MAC keys.

For example, specify the following information on the primary and secondary servers in an HDR pair:

Configuration Parameter	Sample Setting on Primary Server	Sample Setting on Secondary Server
ENCRYPT_HDR	1	1
ENCRYPT_CIPHERS	all	all
ENCRYPT_MAC	medium	medium
ENCRYPT_MACFILE	/vobs/tristan/sqldist/etc/mac1.dat	vobs/tristan/sqldist/etc/mac2.dat
ENCRYPT_SWITCH	60,60	60,60

In this example, the file name in the ENCRYPT_MACFILE path for the primary server is **mac1.dat** and the file name in the ENCRYPT_MACFILE path for the secondary server is **mac2.dat**. Otherwise, all settings are the same on both servers.

Only use these configuration parameters to specify encryption information for HDR. You cannot specify HDR encryption information by using the CSM option in the SQLHOSTS file.

HDR encryption works in conjunction with Enterprise Replication encryption and operates whether Enterprise Replication encryption is enabled or not. When working in conjunction with each other, HDR and Enterprise Replication share the same ENCRYPT_CIPHER, ENCRYPT_MAC, ENCRYPT_MACFILE and ENCRYPT_SWITCH configuration parameters.

For more information on these configuration parameters, see the *IBM Informix Dynamic Server Administrator's Reference*.

Adjusting LRU Flushing and Automatic Tuning in HDR Server Pairs

When a server is configured for HDR, checkpoints triggered by the secondary database server are nonblocking. These types of checkpoints occur infrequently. If a nonblocking checkpoint is triggered by the secondary server, transactions will be blocked on the primary server to make sure that the integrity of the secondary server is not compromised. If nonblocking checkpoints triggered by the secondary server occur on your system, you should tune LRU flushing more aggressively on the primary server to reduce transaction blocking.

To increase LRU flushing, reduce the values of **lru_min_dirty** and **lru_max_dirty** in the BUFFERPOOL configuration parameter.

Automatic LRU tuning can be turned on or off independently on each HDR node. The setting can be different on each HDR database server. For information on turning off automatic LRU tuning, see "Turning Automatic LRU Tuning On or Off" on page 16-6.

For more information on LRU tuning, see the *IBM Informix Dynamic Server Performance Guide*.

Changing the Database Server Mode

To change the database server mode, use the **onmode** utility from the command line or ISA. For information about **onmode**, see the *IBM Informix Dynamic Server Administrator's Reference*.

Table 20-4 summarizes the effects of changing the mode of the primary database server.

Table 20-4. Mode Changes on the Primary Database Server

On the Primary	On the Secondary	To Restart HDR
Any mode to offline (**onmode -k**)	Secondary displays: DR: Receive error. HDR is turned off. The mode remains read-only. If DRAUTO is set to 0 (OFF), the mode remains read-only. If DRAUTO is set to 1 (RETAIN_TYPE), the secondary server switches to standard type and can accept updates. (If DRAUTO is set to 2 (REVERSE_TYPE), the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails.)	Treat it like a failure of the primary. Two different scenarios are possible, depending on what you do with the secondary database server while the primary database server is offline. See these sections for information: • "The Secondary Database Server Was Not Changed to a Standard Database Server" on page 20-28 • "The Secondary Database Server Is Changed to a Standard Database Server Automatically" on page 20-29
to online, quiescent, or administration (**onmode -s** / **onmode -u**) (**onmode -j**)	Secondary does not receive errors. HDR remains on. Mode remains read-only.	Use **onmode -m** on the primary.

Table 20-5 summarizes the effects of changing the mode of the secondary database server.

Table 20-5. Mode Changes on the Secondary Database Server

On the Secondary	On the Primary	To Restart HDR
Read-only offline (**onmode -k**)	Primary displays: DR: Receive error. HDR is turned off.	Treat it as you would a failure of the secondary. Follow the procedures in "Restarting If the Secondary Database Server Fails" on page 20-27.

Note: Single-user mode operates the same way on an HDR secondary database server as it does on the primary database server.

Changing the Database Server Type

You can change the type of either the primary or the secondary database server.

You can change the database server type from secondary to standard only if HDR is turned off on the secondary database server. HDR is turned off when the data replication connection to the primary database server breaks or data replication fails on the secondary database server. When you take the standard database server offline and bring it back online, it does not attempt to connect to the other database server in the replication pair.

Use the following commands to switch the type:

- **hdrmksec.[sh|bat]** and **hdrmkpri.[sh|bat]** scripts

To switch the database server type using hdrmkpri and hdrmksec scripts

1. Shut down the primary database server (**ServerA**):
   ```
   onmode -ky
   ```
2. With the secondary database server (**ServerB**) online, run the **hdrmkpri.sh** script on UNIX or **hdrmkpri.bat** script on Windows. Now **ServerB** is a primary database server.
3. For **ServerA**, run the **hdrmksec.sh** script on UNIX or **hdrmksec.bat** script on Windows. Now **ServerA** is a secondary database server.
4. Bring **ServerB** (primary database server) online.

Monitoring HDR Status

Monitor the HDR status of a database server to determine the following information:

- The database server type (primary, secondary, or standard)
- The name of the other database server in the pair
- Whether HDR is on
- The values of the HDR parameters

Using Command-Line Utilities

The header information displayed every time you execute **onstat** has a field to indicate if a database server is operating as a primary or secondary database server.

The following example shows a header for a database server that is the primary database server in a replication pair, and in online mode:

```
IBM Informix Dynamic Server Version 9.30.UC1   -- online(Prim) -- Up 45:08:57
```

This example shows a database server that is the secondary database server in a replication pair, and in read-only mode.

```
IBM Informix Dynamic Server Version 9.30.UC1     -- Read-Only (Sec) -- Up 45:08:57
```

The following example shows a header for a database server that is not involved in HDR. The type for this database server is standard.

```
IBM Informix Dynamic Server Version 9.30.UC1     -- online -- Up 20:10:57
```

onstat -g dri

To obtain full HDR monitoring information, execute the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)

- The HDR state (on or off)
- The paired database server
- The last HDR checkpoint
- The values of the HDR configuration parameters

For an example of **onstat -g dri** output, see the *IBM Informix Administrator's Reference*.

oncheck -pr

If your database server is running HDR, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that HDR uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in Figure 20-1.

```
Validating Informix Database Server reserved pages - PAGE_1ARCH &
 PAGE_2ARCH
          Using archive page PAGE_1ARCH.

    Archive Level                 0
    Real Time Archive Began       01/11/95 16:54:07
    Time Stamp Archive Began      11913
    Logical Log Unique Id         3
    Logical Log Position          b018

    DR Ckpt Logical Log Id        3
    DR Ckpt Logical Log Pos       80018
    DR Last Logical Log Id        3
    DR Last Logical Log Page      128
```

Figure 20-1. oncheck -pr PAGE_1ARCH Output for Database Server Running HDR

Using SMI Tables

The **sysdri** table, described in the chapter on the **sysmaster** database in the *IBM Informix Administrator's Reference*, contains the following columns.

Column
Description

type HDR server type

state HDR server state

name Database server name

intvl HDR buffer flush interval

timeout
Network timeout

lostfound
HDR lost+found path name

Using ON-Monitor (UNIX)

Choose **Status > Replication** to see information about HDR. This option displays the same information as the **onstat -g dri** option.

HDR Failures

This section discusses the causes and consequences of an HDR failure, as well as the administrator's options for managing failure and restarting data replication.

HDR Failures Defined

An HDR failure is a loss of connection between the database servers in a replication pair. Any of the following situations might cause a data-replication failure:

- A catastrophic failure (such as a fire or large earthquake) at the site of one of the database servers
- A disruption of the networking cables that join the two database servers
- An excessive delay in processing on one of the database servers
- A disk failure on the secondary database server that is not resolved by a mirror chunk

Tip: An *HDR* failure does not necessarily mean that one of the database servers has failed, only that the *HDR* connection between the two database servers is lost.

Detection of HDR Failures

The database server interprets either of the following conditions as an HDR failure:

- A specified time-out value was exceeded.

  During normal HDR operation, a database server expects confirmation of communication from the other database server in the pair. Each database server in the pair has an ONCONFIG parameter, DRTIMEOUT, that specifies a number of seconds. If confirmation from the other database server in a pair does not return within the number of seconds that DRTIMEOUT specifies, the database server assumes that an HDR failure has occurred.
- The other database server in the primary-secondary pair does not respond to the periodic messaging (*pinging*) attempts over the network.

  The database servers ping each other regardless of whether the primary database server sends any records to the secondary database server. If one database server of a primary-secondary pair does not respond to four sequential ping attempts, the other database server assumes that an HDR failure has occurred.

  Each database server in the pair sends a ping to the other database server in the pair when the number of seconds specified by the DRTIMEOUT parameter on that database server has passed.

Actions When an HDR Failure Is Detected

After a database server detects an HDR failure, it writes a message to its message log (for example, `DR: receive error`) and turns data replication off. If an HDR failure occurs, the HDR connection between the two database servers is dropped and the secondary database server remains in read-only mode.

If the secondary database server remains online after a high-availability data-replication failure, and the DRAUTO configuration parameter is set to `1` (RETAIN_TYPE), the type of that database server changes automatically to standard. If DRAUTO is set to `0` (off), the secondary database server periodically attempts to reestablish communication with the primary database server. If

DRAUTO is set to 2 (REVERSE_TYPE), the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails, rather than when the old primary server is restarted.

If Connection Manager is enabled (see Chapter 23, "Manage Cluster Connections with the Connection Manager," on page 23-1), setting DRAUTO to 3 prevents the possibility of having multiple primary servers within a high-availability cluster. If an attempt is made to bring a server on line as a primary server and DRAUTO=3, then Connection Manager Arbitrator will verify that there are no other active primary servers in the cluster. If another primary server is active, then Connection Manager Arbitrator will reject the request.

Considerations After HDR Failure

Consider the following issues when an HDR failure occurs:

- How the clients should react to the failure

  If a real failure occurs (not just transitory network slowness or failure), you probably want clients that are using the failed database server to *redirect* to the other database server in the pair. For instruction on how to redirect clients, see "Redirection and Connectivity for Data-Replication Clients" on page 19-31.
- How the database servers should react to the failure

  Which administrative actions to take after an HDR failure depends on whether the primary database server or the secondary database server failed. For a discussion of this topic, see "Actions to Take If the Secondary Database Server Fails" and "Actions to Take If the Primary Database Server Fails."

  If you redirect clients, consider what sort of load the additional clients place on the remaining database server. You might need to increase the space devoted to the logical log or back up the logical-log files more frequently.

Actions to Take If the Secondary Database Server Fails

If the secondary database server fails, the primary database server remains online.

To redirect clients that use the secondary database server to the primary database server, use any of the methods explained in "Redirection and Connectivity for Data-Replication Clients" on page 19-31. If you redirect these clients, the primary database server might require an additional temporary dbspace for temporary tables and sorting.

You do not need to change the type of the primary database server to standard.

To restart data replication after a failure of the secondary database server, follow the steps in "Restarting If the Secondary Database Server Fails" on page 20-27.

Actions to Take If the Primary Database Server Fails

If the primary database server fails, the secondary database server can operate in the following ways:

- The secondary database server can remain in logical-recovery mode. In this case, no action is taken. This is desirable if you expect the HDR connection to be restored very soon.
- The secondary database server can automatically become a standard database server. This action is called *automatic switchover*.

- The secondary database server can become a standard database server if you use *manual switchover* to change the database server mode to standard.

Automatic Switchover:

Automatic switchover means that the secondary database server automatically becomes a standard database server when DRAUTO=1 or a primary database server when DRAUTO=2 after it detects an HDR failure. It first rolls back any open transactions and then comes into online mode as a primary database server. Automatic switchover occurs only if the parameter DRAUTO in the ONCONFIG file of the secondary database server is set to 1 (RETAIN_TYPE) or 2 (REVERSE_TYPE).

Because the secondary database server becomes a standard or primary database server, you must be sure that either:

- The secondary database server has enough logical-log disk space to allow processing to continue without backing up logical-log files.
- The logical-log files are backed up.

The automatic switchover changes only the type of the database server. It does not redirect client applications to the secondary database server. For information on redirecting clients, see "Redirection and Connectivity for Data-Replication Clients" on page 19-31.

Automatic switchover has the following advantages over manual switchover:

- Clients that you redirect from the primary database server to the secondary database server can continue to write and update data.
- The switchover does not depend on an operator monitoring the message log to see when high-availability data-replication failures occur and then manually switching the secondary database server to a standard database server.

The main disadvantage to automatic switchover is that it requires a very stable network to function appropriately. For more information see "Using Automatic Switchover Without a Reliable Network."

See "The Secondary Database Server Is Changed to a Standard Database Server Automatically" on page 20-29 for the steps required to restart data replication after an automatic switchover.

Actions that Occur After Automatic Switchover:

When you succeed in bringing the original primary database server back online, the HDR connection is automatically established.

- If DRAUTO is set to RETAIN_TYPE, the secondary-turned-standard database server goes through a graceful shutdown (to ensure that all clients that might potentially write to the database server are not connected) and then switches back to a secondary database server.
- If DRAUTO is set to REVERSE_TYPE, the secondary-turned-primary database server switches directly to the primary type. No shutdown occurs. Any applications connected to this database server can stay connected. The original primary database server is switched to a secondary database server.

Using Automatic Switchover Without a Reliable Network:

Although automatic switchover might appear to be the best solution, it is not appropriate for all environments.

Consider what might happen if the primary database server does not actually fail, but appears to the secondary database server to fail. For example, if the secondary database server does not receive responses when it signals (pings) the primary database server because of a slow or unstable network, the secondary server assumes that the primary database server failed and switches automatically to the standard type. If the primary database server also does not receive responses when it signals the secondary database server, it assumes that the secondary database server failed and turns off data replication but remains in online mode. Now the primary database server and the secondary database server (switched to the standard type) are both in online mode.

If clients can update the data on both database servers independently, the database servers in the pair reach a state in which each database server has the logical-log records that are needed by the other. In this situation, you must start again and perform initial data replication with a level-0 dbspace backup of one entire database server, as described in "Starting HDR for the First Time" on page 20-5. Therefore, if your network is not entirely stable, you might not want to use automatic switchover. HDR cannot be reinstated without the risk of losing transactions on the previous secondary server.

Manual Switchover:

Manual switchover means that the administrator of the secondary database server changes the type of the secondary database server to standard. The secondary database server rolls back any open transactions and then comes into online mode as a standard database server, so that it can accept updates from client applications. For an explanation of how to perform the switchover, see "Changing the Database Server Type" on page 20-17.

Restarting After a Manual Switchover:

For a list of the steps involved in restarting data replication after a manual switchover, see "The Secondary Database Server Is Changed to a Standard Database Server" on page 20-28.

Restarting If the Secondary Database Server Is Not Switched to Standard:

If the secondary database server is not changed to type standard, follow the steps in "The Secondary Database Server Was Not Changed to a Standard Database Server" on page 20-28.

Restoring Data After Media Failure Occurs

The result of a disk failure depends on whether the disk failure occurs on the primary or the secondary database server, whether the chunks on the disk contain critical media (the root dbspace, a logical-log file, or the physical log), and whether the chunks are mirrored.

Restoring After a Media Failure on the Primary Database Server

Table 20-6 summarizes the various scenarios for restoring data if the primary database server suffers media failure. The following issues are relevant:

- If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.
- In cases where the chunks are not mirrored, the procedure for restoring the primary database server depends on whether the disk that failed contains critical media.

  If the disk contains critical media, the primary database server fails. You have to perform a full restore using the primary dbspace backups (or the secondary dbspace backups if the secondary database server was switched to standard mode and activity redirected). See "Restarting After Critical Data Is Damaged" on page 20-25.

  If the disk does not contain critical media, you can restore the affected dbspaces individually with a warm restore. A warm restore consists of two parts: first a restore of the failed dbspace from a backup and next a logical restore of all logical-log records written since that dbspace backup. For more information on performing a warm restore, see the *IBM Informix Backup and Restore Guide*. You must back up all logical-log files before you perform the warm restore.

Table 20-6. Scenarios for Media Failure on the Primary Database Server

HDR Server	Critical Media	Chunks Mirrored	Effect of Failure and Procedure for Restoring Media
Primary	Yes	No	Primary database server fails. Follow the procedure in "Restarting After Critical Data Is Damaged" on page 20-25.
Primary	Yes	Yes	Primary database server remains online. Follow the procedures in "Recovering a Mirror Chunk" on page 18-6.
Primary	No	No	Primary database server remains online. Follow the procedure in your Dynamic Server backup and restore manual for performing a warm restore of a dbspace from a dbspace backup. Back up all logical-log files before you perform the warm restore.
Primary	No	Yes	Primary database server remains online. Follow the procedures in "Recovering a Mirror Chunk" on page 18-6.

Restoring After a Media Failure on the Secondary Database Server

High-Availability Data Replication:Table 20-7 on page 20-25 summarizes the various scenarios for restoring data if the secondary database server suffers media failure. The following issues are relevant:

- If chunks are mirrored, you can perform recovery just as you would for a standard database server that uses mirroring.
- In cases where the chunks are not mirrored, the secondary database server fails if the disk contains critical media but remains online if the disk does not contain critical media. In both cases, you have to perform a full restore using the

dbspace backups on the primary database server. (See "Restarting After Critical Data Is Damaged.") In the second case, you cannot restore selected dbspaces from the secondary dbspace backup because they might now deviate from the corresponding dbspaces on the primary database server. You must perform a full restore.

Table 20-7. Scenarios for Media Failure on the Secondary Database Server

HDR Server	Critical Media	Chunks Mirrored	Effect of Failure
Secondary	Yes	No	Secondary database server fails. Primary database server receives errors. HDR is turned off. Follow the procedure in "Restarting After Critical Data Is Damaged."
Secondary	Yes	Yes	Secondary database server remains online in read-only mode. Follow the procedures in "Recovering a Mirror Chunk" on page 18-6.
Secondary	No	No	Secondary database server remains online in read-only mode. Primary database server receives errors. HDR is turned off. Follow the procedure in "Restarting After Critical Data Is Damaged."
Secondary	No	Yes	Secondary database server remains online in read-only mode. Follow the procedures in "Recovering a Mirror Chunk" on page 18-6.

Replicating An Index to the Secondary Server

If an index on an HDR secondary database server becomes corrupt, the normal process for correcting the problem is to drop the index on the primary database server and then rebuild it. This process requires a lock on the table and can take a considerable amount of time to complete. However, instead you can replicate the index from the primary database server to the secondary database server without rebuilding the index on the primary database server.

To manually start index replication, use the **onmode -d** command.

To set up automatic index replication if an index on the secondary database server is detected as corrupt, set the DRIDXAUTO configuration parameter. Use the **onmode -d idxauto** command to update the value of the DRIDXAUTO configuration parameter for the session.

Restarting HDR After a Failure

See "HDR Failures Defined" on page 20-20 for information on the various types of HDR failures. The procedure that you must follow to restart HDR depends on whether or not critical data was damaged on one of the database servers. Both cases are discussed in this section.

Restarting After Critical Data Is Damaged

If one of the database servers experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as if it has no data on the disks, and

you are starting HDR for the first time. Use the functioning database server with the intact disks as the database server with the data.

Critical Media Failure on the Primary Database Server

You might need to restart HDR after the primary database server suffers a critical media failure. Table 20-8 lists the commands required to perform this procedure.

To restart HDR after a critical media failure

1. If the original secondary database server was changed to a standard database server, bring this database server (DRAUTO = 0) to quiescent mode and then use the **onmode -d** command to change the type back to secondary.

   If DRAUTO = 1 (RETAIN_TYPE), this step does not apply. The database server automatically performs a graceful shutdown and switches back to type secondary when you bring the primary database server back online.

   If DRAUTO = 2 (REVERSE_TYPE), the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails, rather than when the old primary server is restarted.
2. Restore the primary database server from the last dbspace backup.
3. Use the **onmode -d** command to set the type of the primary database server and to start HDR.

   The **onmode -d** command starts a logical recovery of the primary database server from the logical-log files on the secondary database server disk. If logical recovery cannot complete because you backed up and freed logical-log files on the original secondary database server, HDR does not start until you perform step 4.
4. Apply the logical-log files from the secondary database server, which were backed up to tape, to the primary database server.

   If this step is required, the primary database server sends a message prompting you to recover the logical-log files from tape. This message appears in the message log. When all the required logical-log files have been recovered from tape, any remaining logical-log files on the secondary disk are recovered.

Table 20-8. Steps for Restarting HDR After a Critical Media Failure on the Primary Database Server

Step	On the Primary Database Server	On the Secondary Database Server
1.		**onmode command** onmode -s onmode -d secondary *prim_name*
2.	**ON–Bar command** onbar -r -p **ontape command** ontape -p	
3.	**onmode command** onmode -d primary *sec_name*	
4.	**ontape command** ontape -l	

Critical Media Failure on the Secondary Database Server

If the secondary database server suffers a critical media failure, you can follow the same steps listed under "Starting HDR for the First Time" on page 20-5.

Critical Media Failure on Both Database Servers

In the unfortunate event that both of the computers that are running database servers in a replication pair experience a failure that damages the root dbspace, the dbspaces that contain logical-log files or the physical log, you need to restart HDR.

To restart HDR after a critical media failure on both database servers

1. Restore the primary database server from the storage space and logical-log backup.
2. After you restore the primary database server, treat the other failed database server as if it had no data on the disks and you were starting HDR

   (See "Starting HDR for the First Time" on page 20-5.) Use the functioning database server with the intact disks as the database server with the data.

Restarting If Critical Data Is Not Damaged

If no damage occurred to critical data on either database server, the following four scenarios, each requiring different procedures for restarting HDR, are possible:

- A network failure occurs.
- The secondary database server fails.
- The primary database server fails, and the secondary database server is not changed to a standard database server.
- The primary database server fails, and the secondary database server is changed to a standard database server.

Restarting After a Network Failure

After a network failure, the primary database server is in online mode, and the secondary database server is in read-only mode. HDR is turned off on both database servers (state = off). When the connection is reestablished, you can restart HDR by issuing **onmode -d secondary** *primary_name* on the secondary database server. Restarting HDR might not be necessary because the primary database server attempts to reconnect every 10 seconds and displays a message regarding the inability to connect every 2 minutes. You do not have to use onmode restart the connection.

Restarting If the Secondary Database Server Fails

If you need to restart HDR after a failure of the secondary database server, complete the steps in Table 20-9 on page 20-28. The steps assume that you have been backing up logical-log files on the primary database server as necessary since the failure of the secondary database server.

Table 20-9. Steps in Restarting After a Failure on the Secondary Database Server

Step	On the Primary	On the Secondary
1.	The primary database server should be in online mode.	oninit If you receive the following message in the message log, continue with step 2: DR: Start Failure recovery from tape
2.		**ontape command** ontape -l **ON–Bar command** onbar -r -l

Restarting If the Primary Database Server Fails

The following sections describe how to restart HDR if the primary database server fails under various circumstances.

The Secondary Database Server Was Not Changed to a Standard Database Server:

If you need to restart HDR after a failure of the primary database server if the secondary database server is not changed to standard, simply bring the primary database server back online using **oninit**.

The Secondary Database Server Is Changed to a Standard Database Server:

If you need to restart HDR after a failure of the primary database server, and you have changed the secondary database server to be a standard database server, complete the steps in Table 20-11 on page 20-30.

Table 20-10. Steps to Restart If You Changed the Secondary Database Server to Standard

Step	On the Primary Database Server	On the Secondary Database Server
1.		onmode -s This step takes the secondary database server (now standard) to quiescent mode. All clients that are connected to this database server must disconnect. Applications that perform updates must be redirected to the primary. See "Redirection and Connectivity for Data-Replication Clients" on page 19-31.
2.		onmode -d secondary *prim_name*

Table 20-10. Steps to Restart If You Changed the Secondary Database Server to Standard (continued)

Step	On the Primary Database Server	On the Secondary Database Server
3.	oninit If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the **oninit** command. If you have backed up and freed the logical-log files on the secondary, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 4). For **ontape** users: If you want to read the logical-log records over the network, set the logical-log tape device to a device on the computer that is running the secondary database server.	
4.	If you are prompted to recover logical-log records from tape, perform this step. **ontape command** ontape -l **ON–Bar command** onbar -r -l	

The Secondary Database Server Is Changed to a Standard Database Server Automatically:

If you need to restart HDR after a failure of the primary database server, and the secondary database server was automatically changed to a standard database server (as described in "Automatic Switchover" on page 20-22), complete the steps shown in the following table.

Table 20-11. Steps to Restart If You Changed the Secondary Database Server to Standard Automatically

Step	On the Primary Database Server	On the Secondary Database Server
1.	`% oninit` If DRAUTO = 1, the type of this database server will be set to primary. If DRAUTO = 2, the type of this database server will be set to secondary when it is restarted. If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the **oninit** command. If logical-log files that you have backed up and freed are on the secondary database server, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 2). For **ontape** users: • Set the logical-log tape device to a device on the computer running the secondary database server.	If DRAUTO = 1, the secondary database server automatically goes through graceful shutdown when you bring the primary back up. This ensures that all clients are disconnected. The type is then switched back to secondary. Any applications that perform updates must be redirected back to the primary database server. See "Redirection and Connectivity for Data-Replication Clients" on page 19-31. If DRAUTO = 2, the secondary database server switches automatically to primary. The old primary database server becomes a secondary database server after it restarts and connects to the other server and determines that it is now a primary database server.
2.	If you are prompted to recover logical-log records from tape, perform this step. **ontape command** `% ontape -l` **ON–Bar command** onbar -r -l	

Chapter 21. Using Remote Standalone Secondary Servers (Enterprise Edition)

In This Chapter

This chapter provides an overview of setting up and configuring RS (remote standalone) secondary servers in a high availability environment.

The chapter discusses the following topics:

- When to use RS secondary servers
- Benefits of RS secondary servers
- How data replicates to an RS secondary server
- RS secondary server security
- Obtaining RS secondary server statistics

RS Secondary Server

An RS (remote standalone) secondary server participates in a high-availability configuration. RS secondary servers can be geographically distant from the primary server, serving as remote backup servers in disaster-recovery scenarios. Each RS secondary server maintains a complete copy of the database, with updates transmitted asynchronously from the primary server over secure network connections.

An RS secondary server provides the ability to configure multiple secondary database servers for increased availability. For example, you can configure one or more RS secondary servers as additional servers to an HDR pair when one or more remote backup sites are needed. While the basic HDR environment provides for a single primary and a single secondary node, an RS secondary server provides for the extension of this environment to support multiple secondary nodes.

Because an RS secondary server maintains a complete copy of the physical database, the RS secondary server must have the same physical storage as the source node.

To aid in disaster recovery scenarios, you can configure RS secondary servers to wait for a specified period of time before applying logs. Delaying the application of log files allows you to recover quickly from erroneous database modifications by restoring from the RS secondary server. See "Delayed Application of Log Records" on page 21-7 for more information.

An RS secondary server replicates all built-in and extended data types. User-defined types (UDTs) must be logged and reside in a single database server. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server. For data stored in an sbspace to be replicated, the sbspace must be logged.

An RS secondary server does not replicate data stored in operating system files or persistent external files or memory objects associated with user-defined routines.

User-defined types, user-defined routines, and DataBlade modules have special installation and registration requirements. For instructions, see “How Data Initially Replicates” on page 19-6.

Comparison of RS Secondary Servers and HDR Secondary Servers

An RS secondary server is similar in many ways to an HDR secondary server. Logs are sent to the RS secondary server in much the same way as a primary server sends logs to an HDR secondary server. However, the RS secondary server is designed to function entirely within an asynchronous communication framework so that its impact on the primary server is minimized. Neither transaction commits nor checkpoints are synchronized between the primary and RS secondary servers. Any transaction committed on the primary server is not guaranteed to be committed at the same time on the RS secondary server.

While an RS secondary server is similar to an HDR secondary server, there are several things that an HDR secondary server supports that an RS secondary server does not support:

- SYNC mode
- DRAUTO parameter
- Synchronized checkpoints

When to use an RS Secondary Server

Use an RS secondary server in your environment for the following reasons:

- Increased server availability

  One or more RS secondary servers provide added assurance by maintaining multiple servers that can be used to increase availability.
- Geographically distant backup support

  It is often desirable to have a secondary server located at some distance from the site for worst-case disaster recovery scenarios. An RS secondary server is an ideal remote backup solution. The high level of coordination between a primary and secondary HDR pair can cause performance issues if the secondary server is located on a WAN (Wide-Area Network). Keeping the primary and secondary servers relatively close together eases maintenance and minimizes performance impact.
- Improved reporting performance

  Multiple secondary servers can offload reporting functionality without impacting the primary server. Also, an RS secondary server configuration makes it easier to isolate reporting requirements from the HA requirements, resulting in better solutions for both environments.
- Availability over unstable networks

  A slow or unstable network environment can cause delays on both the primary and secondary server if checkpoints are achieved synchronously. RS secondary server configurations use fully duplexed networking and require no such coordination. An RS secondary server is an attractive solution if network performance between the primary server and RS secondary server is less than optimal.

Assigning an Alias to an RS Secondary Server

You specify the name of an RS secondary or SD secondary server to a high-availability cluster configuration using the HA_ALIAS configuration parameter.

Figure 21-1 shows entries in an ONCONFIG configuration of an RS Secondary server file that assigns an alias to the secondary server.

```
DBSERVERNAME          reports_srvr
HA_ALIAS              failover_srvr
```

Figure 21-1. Example of DBSERVERNAME and HA_ALIAS Parameters

The alias name is sent to the primary server when a secondary server connects to it. Connection Manager Arbitrator can use this alias name to fail over to a secondary server. If HA_ALIAS is not specified DBSERVERNAME is used. See "HA_ALIAS Configuration Parameter" on page 3-35 for additional information.

Index Page Logging

You must enable index page logging to use an RS secondary server.

How Index Page Logging Works

When an index is created, index page logging writes the pages to the logical log for the purpose of synchronizing index creation between servers in high-availability environments.

Index page logging writes the full index to the log file, which is then transmitted asynchronously to the secondary server. The secondary server can be either an RS secondary or an HDR secondary server. The log file transactions are then read into the database on the secondary server. The secondary server does not need to rebuild the index during recovery. For RS secondary servers, the primary server does not wait for an acknowledgment from the secondary server, which allows immediate access to the index on the primary server.

Control index page logging with the onconfig parameter **LOG_INDEX_BUILDS**. Set the **LOG_INDEX_BUILDS** parameter to 1 (enabled), to build indexes on the primary server and send them to the secondary server.

Enabling or Disabling Index Page Logging

Use the LOG_INDEX_BUILDS configuration parameter to enable or disable index page logging when the database server starts. You can change the value of LOG_INDEX_BUILDS in the onconfig file by running **onmode -wf LOG_INDEX_BUILDS=1** (enable) or **0** (disable).

Index page logging must be enabled when an RS secondary server exists in a high-availability environment.

Viewing Index Page Logging Statistics

You can use the onstat utility or system-monitoring interface (SMI) tables to view whether index page logging is enabled or disabled. The statistics also display the date and time index page logging was enabled or disabled.

To view index page logging statistics, use the **onstat -g ipl** command, or query the **sysipl** table.

For an example of **onstat -g ipl** output, see information on the onstat utility in the *IBM Informix Administrator's Reference*.

Server Multiplexer Group (SMX) Connections

Server Multiplexer Group (SMX) is a communications interface that supports encrypted multiplexed network connections between servers in high availability environments. SMX provides a reliable, secure, high-performance communication mechanism between database server instances.

Enabling SMX Encryption

Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high availability configurations. If you set the ENCRYPT_SMX parameter to 1, encryption is used for SMX transactions only when the database server being connected to also supports encryption. If you set the ENCRYPT_SMX configuration parameter to `2` , only connections to encrypted database servers are allowed. Setting ENCRYPT_SMX to `0` disables encryption between servers.

Obtaining SMX Statistics

You can use the onstat utility or system-monitoring interface (SMI) tables to view SMX connection statistics or SMX session statistics.

To view SMX connection statistics, use the **onstat -g smx** command.

To view SMX session statistics, use the **onstat -g smx ses** command.

For examples of **onstat -g smx** and **onstat -g smx ses** output, see information on the onstat utility in the *IBM Informix Administrator's Reference*.

Configuring an RS Secondary Server

This section describes the hardware requirements for configuring RS secondary server environments.

Hardware and Software Requirements

The RS secondary server maintains a complete copy of the physical database. Because of this, the following must be identical to the primary server:

- The computer hardware running the database server
- The amount of disk space allocated to dbspaces
- The offsets into the physical devices used in creating the dbspaces

For a complete list of hardware and operating system requirements for secondary servers, see "Configuring a System for HDR" on page 20-1.

Starting an RS Secondary Server for the First Time

After you complete the hardware configuration of the RS secondary server, you are ready to start the RS secondary server. This section describes the necessary steps for starting an RS secondary server and connecting it to a primary server.

Suppose you want to start a primary server and an RS secondary server, **ServerA** and **ServerB**. The procedure for starting the servers, using **ServerA** as the primary

database server and **ServerB** as the RS secondary database server, is described in the following steps. Table 21-1 on page 21-6 lists the commands required to perform each step.

The procedure requires that the primary server be backed up and then restored onto the secondary server. You can use **ontape** or ON–Bar to perform the backup and restore. You must use the same utility throughout the procedure.

Important: If you use ON–Bar to perform the backup and restore, **ontape** is required on both database servers. You cannot remove **ontape** from database servers participating in an RS secondary server configuration.
You can also set up an RS secondary server using standard ON–Bar or **ontape** commands for external backup and restore.

To start a primary server with an RS secondary server

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.

   For information on how to install user-defined types or user-defined routines see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*. For information on how to install DataBlade modules, see the *IBM Informix DataBlade Module Installation and Registration Guide*.
2. Activate index page logging on the primary server.
3. Record the identity of the RS secondary server on the primary server. The password is optional and should be specified only the first time this command is issued. Attempting to change the password after the RS secondary server has connected will result in an error.
4. Create a level-0 backup of **ServerA**.
5. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 4. Do not perform a logical restore.

   Use the appropriate command:

   - Use the **onbar -r -p** command to perform a physical restore.
   - Use the **onbar -r -p -e** command to perform a physical external restore.
   - Use the **ontape -p** option. (Do not use the **ontape -r** option because it performs both a physical and a logical restore.)
   - Use the **ontape -p -e** command to perform the physical external restore.
6. Use the **onmode -d RSS ServerA password** command to set the type of **ServerB** to an RS secondary server and indicate the associated primary database server.

   ServerB tries to establish a connection with the primary database server (**ServerA**) and start operation. The connection should be successfully established.

   The secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 4. If all these logical-log records still reside on the primary database server disk, the primary database server sends these records directly to the RS secondary server over the network and logical recovery occurs automatically.

   If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 7 on page 21-6.

Important: You must complete steps 5 on page 21-5 through 6 on page 21-5 during the same session. If you need to shut down and restart the secondary database server after step 5 on page 21-5, you must redo step 5 on page 21-5.

7. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.

   If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

   After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

Table 21-1. Steps to Start a Primary with an RS secondary server for the First Time

Step	On the Primary	On the RS Secondary
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**onmode command** onmode -wf LOG_INDEX_BUILDS=1	
3.	**onmode command** onmode -d add RSS *rss_servername password*	
4.	**ontape command** ontape -s -L 0 **ON–Bar command** onbar -b -L 0	
5.		**ontape command** ontape -p or ontape -p -e Answer `no` when you are prompted to back up the logs. **ON–Bar command** onbar -r -p or onbar -r -p -e
6.		**onmode command** onmode -d RSS *primary_servername password* If all the logical-log records written to the primary database server since step 1 still reside on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 7 must be performed).

Table 21-1. Steps to Start a Primary with an RS secondary server for the First Time (continued)

Step	On the Primary	On the RS Secondary
7.		**ontape command** ontape -l **ON–Bar command** onbar -r -l This step is needed only when the secondary database server prompts you to recover the logical-log files from the tape backup.

Decreasing Setup Time Through an Alternate Backup Method:

You can dramatically improve the speed of setting up a secondary server by using the **ontape** `STDIO` feature. See "Decreasing Setup Time Using the ontape STDIO Feature" on page 20-8 for more information.

See the *IBM Informix Backup and Restore Guide* for details about using the `STDIO` value.

Delayed Application of Log Records

To aid in disaster recovery scenarios, you can configure RS secondary servers to wait for a specified period of time before applying logs received from the primary server.

By delaying the application of log files you can recover quickly from erroneous database modifications by restoring the database from the RS secondary server. You can also stop the application of logs on an RS secondary server at a specified time.

For example, suppose a database administrator wants to delete certain rows from a table based on the age of the row. Each row in the table contains a timestamp that indicates when the row was created. If the database administrator inadvertently sets the filter to the wrong date, more rows than intended might be deleted. By delaying the application of log files, the rows would still exist on the RS secondary server. The database administrator can then extract the rows from the secondary server and insert them on the primary server.

Now suppose a database administrator needs to perform changes to the schema by renaming a table, but types the wrong command and drops the table **orders** instead of changing the table name to **store_orders**. If an RS secondary server is configured to delay application of logs, the database administrator can recover the **orders** table from the secondary server.

When delayed application of log files configured, transactions sent from the primary server are not applied until after a specified period of time has elapsed. Log files received from the primary server are staged in a specified secure directory on the RS secondary server, and then applied after the specified period of time. There are two ways to delay the application of log files:

- Apply the staged log files after a specified time interval
- Stop applying log files at a specified time

You enable the delayed application of log files by setting configuration parameters in the onconfig file of the RS secondary server. You must specify the directory in which log files are staged by setting the LOG_STAGING_DIR configuration parameter before enabling the delayed application of log files. After specifying the LOG_STAGING_DIR configuration parameter, you configure the DELAY_APPLY or STOP_APPLY configuration parameters either by editing the onconfig file or dynamically using **onmode -wf** commands.

Where Log Records are Stored

The server creates additional directories named **ifmxlog_##** in the directory specified by LOG_STAGING_DIR, where **##** is the instance specified by SERVERNUM. The directories are used to store the logical logs and are also used during the recovery of the RS secondary server. If recovery of the RS secondary server becomes necessary, and the logs have wrapped on the primary server, then the logs in **ifmxlog_##** can be used to recover the server. The files within **ifmxlog_##** are purged when no longer needed.

Conditions that Trigger Delays

The time values in the BEGIN WORK, COMMIT WORK, and ROLLBACK WORK log records are used to calculate how to delay or stop the application of log files. The time values are calculated before passing the log pages to the recovery process.

When a BEGIN WORK statement is issued, the BEGIN WORK log record is not written until the first update activity is performed by the transaction; therefore, there can be a delay between the time that the BEGIN WORK statement is issued and when the BEGIN WORK log is written.

Interaction with Secondary Server Updates

You should consider the interaction between secondary server updates and delayed application of log files. If updates are enabled, and the secondary server is updated, the updates are not applied until after the amount of time specified by DELAY_APPLY. Disabling secondary server updates, however, also disables Committed Read, which guarantees that every retrieved row is committed in the table at the time that the row is retrieved.

To retain the Committed Read isolation level, consider enabling secondary server updates using the UPDATABLE_SECONDARY configuration parameter, but removing the RS secondary server used for delayed application of log files from the Connection Manager service-level agreement list. Alternatively, consider moving the RS secondary server to a new SLA.

See "Update Data on Secondary Servers" on page 19-11 and *IBM Informix Dynamic Server Administrator's Reference* for more information.

Specifying the Log Staging Directory

You configure the log staging directory to specify where log files on RS secondary servers are staged before being applied to the database.

You must specify a staging directory for log files sent from the primary server before enabling delayed application of log files. No default staging directory is defined. The server creates additional directories in the directory specified by LOG_STAGING_DIR named **ifmxlog_##**, where **##** is the instance specified by

SERVERNUM. The directories are used to store the logical logs and are also used during the recovery of the RS secondary server. The staged log files are automatically removed when they are no longer needed. If the files within LOG_STAGING_DIR are lost, and the primary server has overwritten the logs, then the RS secondary server must be rebuilt.

You should ensure that the directory specified by LOG_STAGING_DIR exists and is secure. The directory must be owned by user informix, must belong to group informix, and must not have public read or write permission. If role separation is enabled, the directory specified by LOG_STAGING_DIR must be owned by the user or group that owns $INFORMIXDIR/etc. If the directory specified by LOG_STAGING_DIR is not secure, then the server cannot be initialized. The following message is written to the online message log if the directory is not secure:

```
The delay apply directory (directory_name) is not secure.
```

You should also ensure that the disk contains sufficient space to hold all of the logs from the primary server, and that the directory does not contain staged logs from previous instances that are no longer being used.

See *IBM Informix Dynamic Server Administrator's Reference* for more information.

To set LOG_STAGING_DIR:

1. Ensure that the directory in which logs are to be stored exists and is secure.
2. Edit the RS secondary server onconfig file.
3. Specify the staging directory as follows: `LOG_STAGING_DIR directory_name` *where directory_name* is the name of the directory in which to store the logs.
4. Restart the server.

You can also set the LOG_STAGING_DIR configuration parameter without having to restart the server by using the **onmode -wf** command; however, the delayed application of log files must not be active when the command is run.

Delaying Application of Log Records on an RS Secondary Server

You can delay the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

You enable the delayed application of log files by setting the DELAY_APPLY configuration parameter. You can manually edit the **onconfig** file and restart the server, or you can change the value dynamically using the **onmode -wf** command. When setting the value of DELAY_APPLY you must also set LOG_STAGING_DIR. If DELAY_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

Set DELAY_APPLY using both a *number* and a *modifier*. *Number* may contain up to three digits and indicates the number of modifier units. *Modifier* is one of:

- D (or d) for days
- H (or h) for hours
- M (or m) for minutes
- S (or s) for seconds

See *IBM Informix Dynamic Server Administrator's Reference* for more information.

To delay the application of log files on the RS secondary for four hours:

+ Run the following command: `onmode –wf DELAY_APPLY=4H`

+ To delay the application of log files for one day:
+ `onmode -wf DELAY_APPLY=1D`

+ To disable delayed application of log files:
+ `onmode –wf DELAY_APPLY=0`

+ ### Stopping the Application of Log Records
+ You can halt the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

+ You stop the application of log files on the RS secondary server by setting the STOP_APPLY configuration parameter. You can manually edit the **onconfig** file and restart the server, or you can change the value dynamically using the **onmode -wf** command. When setting the value of STOP_APPLY you must also set LOG_STAGING_DIR. If STOP_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

+ See *IBM Informix Dynamic Server Administrator's Reference* for more information.

+ To stop the application of log files on the RS secondary server immediately:

+ Run the following command: `onmode –wf STOP_APPLY=1`

+ To stop the application of log files at 11:00 p.m. on April 15th, 2009:
+ `onmode –wf STOP_APPLY="2009:04:15-23:00:00"`

+ To resume the normal application of log files
+ `onmode –wf STOP_APPLY=0`

+ Obtaining RS Secondary Server Statistics

Use the **onstat** command to display information about the state of RS secondary servers. To display the number of RS secondary servers, information about the data that has been sent to the RS secondary servers, and information about what the RS secondary servers have acknowledged receiving, use the following command:

```
onstat -g rss
```

This command has options to show extended information about a single server or multiple secondary servers. For examples of **onstat -g rss** output, see information on the **onstat** utility in the *IBM Informix Administrator's Reference*.

Removing an RS Secondary Server

Remove an RS secondary server from a high-availability cluster by issuing the following command:

```
onmode -d delete RSS rss_servername
```

RS Secondary Server Security

RS secondary servers support similar encryption rules as HDR. See "Encrypting Data Traffic Between HDR Database Servers" on page 20-15 for details.

See “Server Multiplexer Group (SMX) Connections” on page 21-4 for additional information on setting up and configuring encryption between servers and RS secondary servers.

Creating or Changing a Password on an RS Secondary Server

The identity of each RS secondary server must be defined on the primary server in order to prevent unauthorized servers from connecting to the network and obtaining log information. The name and password of each RS secondary server can be defined either before or after the level-0 backup of the primary server. This is done by issuing this command:

```
onmode -d add RSS rss_servername [optional password]
```

You can change the password only if the server is not connected to a high availability environment. Attempting to change the password of an RS secondary server that is already connected will return an error. To change the password on an RS secondary server before it is connected use the following command on the primary server:

```
onmode -d change RSS rss_servername password
```

Chapter 22. Using Shared Disk Secondary Servers (Enterprise Edition)

In This Chapter

This chapter provides an overview of setting up and configuring SD (shared disk) secondary servers in a high-availability environment. SD secondary server options are available with the standard version of Dynamic Server. Many high availability options do not work with the IBM Informix Dynamic Server Express Edition.

The chapter discusses the following topics:

- When to use SD secondary servers
- Hardware and Software Requirements for SD Secondary Servers
- Setting up an SD Secondary Server
- Obtaining SD Secondary Server Statistics
- SD Secondary Server Configuration
- SD Secondary Server Security

SD secondary servers are not supported in Windows environments.

SD Secondary Server

An SD secondary server participates in high-availability cluster configurations. In such configurations, the primary server and the SD secondary server share the same disk or disk array. An SD secondary server does not maintain a copy of the physical database on its own disk space; rather, it shares disks with the primary server.

SD secondary servers must be configured to access shared disk devices that allow concurrent access. Do not configure an SD secondary server that uses operating system buffering, such as NFS cross-mounted file systems. If the SD secondary server instance and the primary server instance both reside on a single machine, then both servers can access local disks. If the SD secondary server and the primary server are on separate physical machines, then they must be configured to access shared disk devices that appear locally attached, such as Veritas or GPFS™.

SD secondary servers can be used in conjunction with HDR secondary servers, RS secondary servers, and with Enterprise Replication.

SD secondary servers can be added to a high availability environment very quickly because they do not require a separate copy of the disk. Since the SD server shares the primary server's disk, it is recommended that you provide some other means of disk back such as disk mirroring or the use of an RS secondary server or an HDR secondary server.

An SD secondary server cannot be promoted to an RS secondary server. An SD secondary server cannot be promoted to a standard server that would exist outside the primary high availability environment.

When to Use an SD Secondary Server

Use an SD secondary server for these reasons:

- Increased reporting capacity

  Multiple secondary servers can offload reporting functionality without impacting the primary server.

- Server failure backup

  In the event of a failure of the primary server, an SD secondary can be promoted quickly and easily to a primary server. For example, if you are using SAN (storage area network) devices that provide ample and reliable disk storage but you are concerned with server failure, SD secondary servers can provide a reliable backup.

Hardware and Software Requirements for SD Secondary Servers

Except for disk requirements (which are shared with the primary server), hardware and software requirements are generally the same as for HDR secondary servers (Please refer to the Machine Notes for specific supported platforms). In addition, the primary disk system must be shared across the computers that are hosting the database servers. This means that the path to the dbspaces from the SD secondary is the same dbspace path as the primary server. Refer to: "Configuring a System for HDR" on page 20-1.

Setting up an SD Secondary Server

An SD secondary server is set up by first setting the **SDS_TIMEOUT** configuration parameter. Next, the **onmode** utility is used to set the primary server alias that the SD secondary server will use to connect to the primary server. Then, the configuration file on the SD secondary is modified to include the appropriate options. Finally, the **oninit** utility is run to start the SD secondary server.

1. On the primary server, set the **SDS_TIMEOUT** configuration parameter in the onconfig file:

   ```
   SDS_TIMEOUT x
   ```

   SDS_TIMEOUT specifies the amount of time in seconds that the primary server will wait for a log position acknowledgment to be sent from the SD secondary server. See the *IBM Informix Administrator's Reference* for information on the **SDS_TIMEOUT** configuration parameter.

2. On the primary server, configure the alias name of the SD primary server:

   ```
   onmode -d set SDS primary <alias>
   ```

   The server name specified by *<alias>* will become the primary server of the shared disk environment and the source of logs for the SD secondary server.

3. On the SD secondary server set the following configuration parameters in the configuration file:

   ```
   SDS_ENABLE 1
   SDS_PAGING <path 1>,<path 2>
   SDS_TEMPDBS <dbspace_name>,<path>,<pagesize>,<offset>,<offset_size>
   ```

   SDS_ENABLE must be set to 1 (enable) on the secondary server to enable support of the shared disk environment. **SDS_PAGING** specifies the path to two files that are used to hold pages that may need to be flushed between checkpoints. Each file acts as temporary disk storage for chunks of any page size. **SDS_TEMPDBS** is used to define the temporary dbspace used by the SD

secondary server. This dbspace is dynamically created when the server is started (it is not created by running **onspaces**). See the *IBM Informix Administrator's Reference* for additional information on these parameters.

4. On the SD secondary server, set the following configuration parameters to match those on the primary server:
   - ROOTNAME
   - ROOTPATH
   - ROOTOFFSET
   - ROOTSIZE
   - PHYSFILE
   - LOGFILES
   - LOGSIZE

   Map the other configuration parameters to match those of the primary server with the exception of **DBSERVERALIASES**, **DBSERVERNAME**, and **SERVERNUM**.
5. Add an entry to the **sqlhosts** file (or for Windows systems, the SQLHOSTS registry key) for the primary server:

   ```
   primary_dbservername  nettype  primary_hostname  servicename
   ```
6. Start the SD secondary server using the **oninit** command.

   ```
   oninit
   ```

   The primary server must be active before starting the SD secondary server.

   When a secondary server is started, it must first process any open transactions using fast recovery mode. Client applications can connect to the server only after all of the transactions open at the startup checkpoint have either committed or rolled back. After open transactions have been processed, client applications can connect to the server as they normally do. You should examine the online.log file on the secondary server to verify that it has completed processing open transactions.

Table 22-1 illustrates the preceding steps so that you can clearly determine which steps are performed on the primary server and which are performed on the secondary server.

Table 22-1. Steps to Start an SD secondary server for the First Time

Step	On the Primary	On the Secondary
1.	Set the SDS_TIMEOUT configuration parameter in the onconfig file: `SDS_TIMEOUT x`	
2.	Configure the alias name of the SD primary server: `onmode -d set SDS primary <alias>`	

3		Set configuration parameters:
		`SDS_ENABLE 1`
		`SDS_PAGING <path 1>,<path 2>`
		`SDS_TEMPDBS <dbspace_name>,<path>,<pagesize>,`
		`<offset>,<offset_size>`

Table 22-1. Steps to Start an SD secondary server for the First Time (continued)

Step	On the Primary	On the Secondary
4		Set configuration parameters to match those on the primary server: • ROOTNAME • ROOTPATH • ROOTOFFSET • ROOTSIZE • PHYSFILE • LOGFILES • LOGSIZE
5		Add an entry to the **sqlhosts** file (or for Windows systems, the SQLHOSTS registry key) for the primary server: *dbservername nettype hostname servicename*
6		Start the SD secondary server `oninit`

There will be increased memory usage in the LGR memory pool when a secondary server is added.

See the *IBM Informix Administrator's Reference* for information on configuration parameters.

Assigning an Alias to an SD Secondary Server

You specify the name of an RS secondary or SD secondary server to a high-availability cluster configuration using the HA_ALIAS configuration parameter.

Figure 22-1 shows entries in an ONCONFIG configuration of an SD Secondary server file that assigns an alias to the secondary server.

```
DBSERVERNAME          reports_srvr
HA_ALIAS              failover_srvr
```

Figure 22-1. Example of DBSERVERNAME and HA_ALIAS Parameters

The alias name is sent to the primary server when a secondary server connects to it. Connection Manager Arbitrator can use this alias name to fail over to a secondary server. If HA_ALIAS is not specified DBSERVERNAME is used. See "HA_ALIAS Configuration Parameter" on page 3-35 for additional information.

Obtaining SD Secondary Server Statistics

Use the **onstat** utility or system-monitoring interface (SMI) tables to view SD secondary server statistics.

Use **onstat -g sds** to view SD secondary server statistics. The output of the onstat utility depends on whether the utility is run on the primary or secondary server.

Query the **syssrcsds** table to obtain information on shared disk statistics on the primary server.

Query the **systrgsds** table to obtain information on shared disk statistics on the secondary server.

For information on **onstat** and SMI tables see the *IBM Informix Administrator's Reference*.

SD Secondary Server Configuration

Promoting an SD Secondary Server to a Primary Server

Convert an SD secondary server to a primary server by issuing the following command on the SD secondary server:

```
onmode -d set SDS primary <alias>
```

An SD secondary server cannot be converted to a standard server.

Converting a Primary Server to a Standard Server

You can convert a primary server to a standard server and disconnect it from the shared disk environment using the following command on the primary server:

```
onmode -d clear SDS primary <alias>
```

SD Secondary Server Security

SD secondary servers support similar encryption rules as HDR. See "Meeting Database Server Configuration Requirements" on page 20-3 for details.

Encryption can be enabled or disabled between any primary and secondary server pair. That is, you can encrypt traffic between the primary server and one SD secondary server and not encrypt traffic between the primary server and another SD secondary server.

See the "Server Multiplexer Group (SMX) Connections" on page 21-4 section for additional information on setting up and configuring encryption between primary servers and SD secondary servers.

Chapter 23. Manage Cluster Connections with the Connection Manager

The Connection Manager is a utility that monitors and maintains connections in a high-availability cluster, and directs connection requests from client applications to the appropriate server in the cluster.

Because high-availability clusters (also called MACH11 clusters) often consist of multiple database servers, client applications can connect to the primary server or to one of many secondary servers. With a large numbers of servers, it can be difficult to determine which server to connect to. It can be also be difficult to determine which server has sufficient free resources to perform a given task. Finally, it is difficult (if not impossible) to know when a server may encounter a problem. You use the Connection Manager to solve these difficulties.

The Connection Manager can balance workloads by directing client application connections to the server with the least amount of activity. The Connection Manager utility also performs failover arbitration. You can configure Connection Manager to ensure that, if a primary server fails, another server in the high-availability cluster will automatically take over the role of the primary server.

The Connection Manager performs three roles:

- Rule-based connection redirection
- Cluster load balancing
- Cluster failover

Rule-based Connection Redirection

Applications connect to the Connection Manager as if they are connecting to the actual database server. When applications connect to Connection Manager, redirection of the connection takes place in the communication layer and no additional action is required by the application.

In order to configure the Connection Manager, the database administrator must set up a daemon (or on Windows systems, a service) called **oncmsm** with customized redirection rules called Service Level Agreements (SLAs). Once the Connection Manager is configured and initialized, it accepts connection requests from client applications and redirects them to the appropriate server based on the SLA (redirection rule).

Cluster Load Balancing

The Connection Manager can perform configurable load balancing, in which redirection is based on server work load. Connection Manager connects to each of the servers in the cluster and gathers statistics regarding the type of server, unused workload capacity, and the current state of the server. From this information, the Connection Manager is able to redirect the client connection to the server with the least amount of activity. Load balancing is configured using service level agreements.

Cluster Failover

You can configure automatic failover with the Connection Manager. If the Connection Manager detects that the primary server has failed, and no action is taken by the primary server to re-connect during the ensuing timeout period, then the most appropriate secondary server is converted to the primary server. You configure Connection Manager failover parameters using a configuration file.

Note: The Connection Manager both monitors the servers within a high-availability cluster and it helps client applications connect to the most appropriate server. These roles, however, are separate; once the Connection Manager connects a client to a server, it will not redirect the client thereafter. If a database server to which an application is connected experiences a problem, the application must request a connection through the Connection Manager again.

The Connection Manager supports Distributed Relational Database Architecture (DRDA) connections. For more information, see "Overview of DRDA" on page 3-36 and "Configuring Dynamic Server for Connections to IBM Data Server Clients" on page 3-37.

Installing the Connection Manager

You install the Connection Manager to dynamically route client application connection requests and to provide failover capabilities for high-availability clusters.

The Connection Manager is packaged with the IBM Informix Client SDK (CSDK) version 3.50 and higher. Before you configure and use the Connection Manager, you must install the Client SDK. See the *IBM Informix Client Products Installation Guide*.

To improve performance, you should install the Connection Manager on the computers on which the database applications are running.

Setting up and Starting the Connection Manager

To configure the Connection Manager, you must set up an encrypted password file, service level agreements, failover parameters, and configure the sqlhosts file.

Prerequisites:

UNIX Only: Only user **informix** can execute **oncmsm**. If user **root** or a member of the DBSA group is granted privileges to connect to the sysadmin database, then user **root** or that member of the DBSA group can also invoke **oncmsm**.

Windows Only: You must be a member of the **Informix-Admin** group to execute **oncmsm**. If user **administrator** or a member of the DBSA group is granted privileges to connect to the sysadmin database, then user **administrator** or that member of the DBSA group can also invoke **oncmsm**.

To configure and start the Connection Manager:

1. Create and encrypt a password file (see "Creating an Encrypted Password File" on page 23-3).
2. Set the INFORMIXSERVER and INFORMIXDIR environment variables (see "Setting the Environment Variables" on page 23-4).

3. Create a Connection Manager configuration file on the machine on which the Connection Manager is to run. The configuration file defines the Connection Manager instance name, the service level agreements, debug parameter, the name of the log file, and the failover configuration. See "Connection Manager Setup Example" on page 23-5 for an example, or see *IBM Informix Dynamic Server Administrator's Reference* for the configuration file layout.
4. Create entries for the primary server and all SLAs by editing the **sqlhosts** file on UNIX, or by editing the SQLHOST registry key with the **setnet32** utility on Windows (see "Modifying the sqlhosts file" on page 23-4).
5. Use the **oncmsm** utility to start the Connection Manager.

   UNIX Only: Start the Connection Manager on UNIX systems with the following command:

   ```
   oncmsm -c configuration_file
   ```

   Windows Only: Starting the Connection Manager on Windows systems is a two-step process. You install the service first then start the Connection Manager. To install the Connection Manager service:

   ```
   oncmsm -i -c configuration_file
   ```

   To start the Connection Manager:

   ```
   oncmsm Connection_Manager_Name
   ```

   Connection_Manager_Name is the name of the Connection Manager instance.

   The following message is displayed when starting the Connection Manager on Windows systems:

   ```
   Please specify the user and password to run this service.
   Press <ENTER>  to run Connection Manager as 'localsystem'
   ```

   Either enter the informix user ID and password, or press Enter to automatically create an informix-admin group and assign it the necessary access rights.

Creating an Encrypted Password File

If any of the servers within the cluster are outside of a trusted network environment, you should create an encrypted password file so that the Connection Manager can create secure connections to each of the servers in the cluster.

You use the password management utility, **onpassword**, to either encrypt or decrypt a password file.

To create an encrypted password file:

1. Use a text editor to create an ASCII text file containing the server names, user names, and passwords for all servers in the cluster.
2. Determine a key that will be used to encrypt the file. The key can consist of any sequence of characters or numbers, but should not contain spaces. In order to later decrypt the password file, you must supply the same key that was used to encrypt the file.
3. Encrypt the password file with the **onpassword** utility, specifying the key name and the password file name:

   ```
   onpassword -k secretkey -e ./password_file
   ```

See the *IBM Informix Dynamic Server Administrator's Reference* for additional information about the **onpassword** utility.

Modifying an Encrypted Password File

You might want to modify the encrypted password file if you add or remove servers from your cluster, change the passwords, or change your secret key.

To edit the encrypted password file:

1. Decrypt the password file with the **onpassword** utility, specifying the secret key name and the password file name:

   ```
   onpassword -k secretkey -d ./password_file
   ```

2. If necessary, edit the file with a text editor to make the required changes.
3. If necessary, determine a new secret key. The key can consist of any sequence of characters or numbers, but should not contain spaces. In order to later decrypt the password file, you must supply the same key that was used to encrypt the file.
4. Encrypt the password file with the **onpassword** utility, specifying the secret key name and the password file name:

   ```
   onpassword -k secretkey -e ./password_file
   ```

Setting the Environment Variables

The INFORMIXDIR environment variable should point to the directory in which the CSDK was installed and INFORMIXSERVER should contain the name of the primary database server in the cluster.

If you are using the UNIX C shell (csh), use the **setenv** command to set the environment variables. For other shells, use the method appropriate for that shell.

```
setenv INFORMIXDIR path_to_csdk
setenv INFORMIXSERVER name_of_primary_server
```

To use a file other than **sqlhosts** to specify Connection Manager settings, set the **INFORMIXSQLHOSTS** environment variable to the name of that file.

For Windows systems, use the **Environment** tab of the **setnet32** utility to set or verify environment variables.

Modifying the sqlhosts file

Modify the sqlhosts file to define network connectivity information.

The Connection Manager is configured using the **sqlhosts** file in the same way that any server is configured; however, each entry in the sqlhosts file for the Connection Manager represents a service level agreement (SLA) name and not a physical server.

1. Edit the sqlhosts file on the server on which Connection Manager is to run.
2. Add a line containing the name, net type, host name and service name of the primary server.
3. Add one line for each of the service level agreement names. For example, the following lines create service level agreement names of **oltp** and **report**:

   ```
   oltp     onsoctcp      cmhost1 cmport1
   report   ontlitcp      cmhost1 cmport3
   ```

   In the example, **oltp** represents the name of the Service Level Agreement (SLA). The SLA name is what clients use to connect to a server through the Connection Manager. **cmhost1** represents the name of the server on which Connection Manager is running. **cmport1** and **cmport3** are the ports to which clients connect.

You should also include a line in the sqlhosts file containing connectivity information for the primary server.

See "The sqlhosts File and the SQLHOSTS Registry Key" on page 3-13 for more information.

Connection Manager Setup Example

This example shows how to set up the Connection Manager for a small high-availability cluster.

Suppose you had a configuration consisting of three servers: a primary server, an HDR secondary server, and an SD (shared disk) secondary server. You want the cluster to provide three different services: OLTP (On-Line Transaction Processing), PAYROLL, and REPORTING to support different applications. You decide that all OLTP services can be run on the primary server; PAYROLL services can be run on either the HDR secondary or the primary server; and REPORT services can be run on the HDR secondary or on the shared disk secondary servers.

Figure Figure 23-1 illustrates the desired configuration. The illustration shows client applications issuing connection requests to the Connection Manager. Based on predefined service level agreements, the Connection Manager routes the connections to the appropriate server.

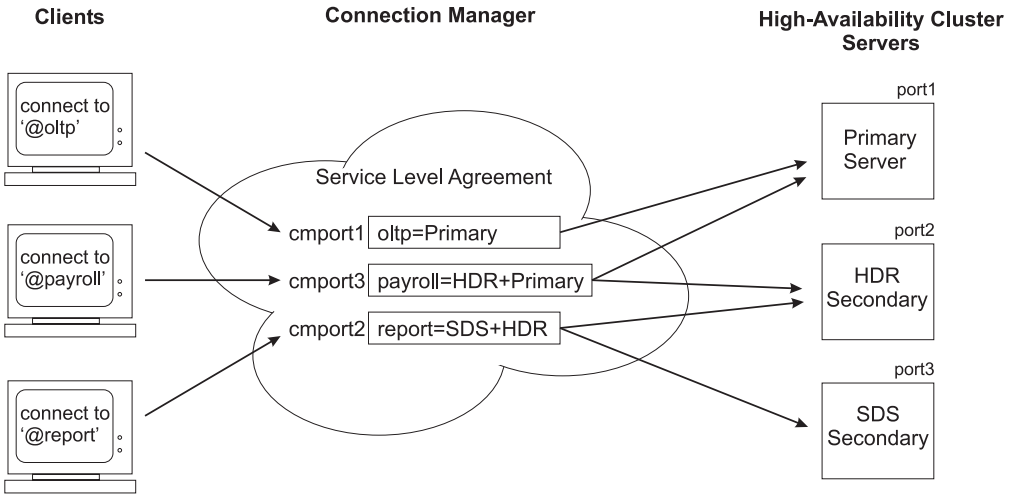


Figure 23-1. Connection Manager Configuration

To configure and start the Connection Manager:

1. Create and encrypt a password file. For this example, create a file called `passwords.txt` containing the following:

```
ids      -     informix password1
ids_hdr  -     informix password2
ids_sds  -     informix password3
```

Encrypt the file using the command:

```
onpassword -k SecretKey -e ./passwords.txt
```

2. Set the INFORMIXSERVER and INFORMIXDIR environment variables (see "Setting the Environment Variables" on page 23-4.
3. Create the Connection Manager configuration file. For this example, on the machine on which Connection Manager is to run, create a file called **cmconfig** that contains the following lines:

```
NAME cm_example
SLA oltp=primary
SLA payroll=HDR+primary
SLA report=SDS+HDR
DEBUG 0
LOGFILE cmlog
FOC ids_sds+ids_hdr,10
```

   The configuration file names the Connection Manager instance **cm_example**, describes the service level agreements, turns debug off, defines the name of the log file, and defines a failover configuration.
4. Edit the primary database server's **sqlhosts** file (or for Windows systems, the SQLHOSTS registry key) as follows:

```
ids       onsoctcp host1  port1
ids_hdr   onsoctcp host2  port2
ids_sds   onsoctcp host3  port3
```

5. On the machine on which the Connection Manager will run, and on each of the client machines, edit the **sqlhosts** file (or for Windows systems, the SQLHOSTS registry key) as follows:

```
ids       onsoctcp  host1    port1
oltp      onsoctcp  cmhost1  cmport1
report    onsoctcp  cmhost1  cmport2
payroll   onsoctcp  cmhost1  cmport3
```

6. Use the **oncmsm** utility to start the Connection Manager.

```
oncmsm -c cmconfig
```

7. Examine the log file to verify that Connection Manager started correctly. For example, the following shows the log file for a successful Windows installation:

```
13:18:07 IBM Informix Connection Manager
13:18:07 Connection Manager name is cm_example
13:18:07 Starting Connection Manager...
13:18:07 dbservername = ol_ids_1150_2
13:18:07 nettype      = olsoctcp
13:18:07 hostname     = *IBM-46560C7C770
13:18:07 servicename  = svc_ids_1150_2
13:18:07 options      =
13:18:07 listener oltp initializing
13:18:07 listener payroll initializing
13:18:07 listener report initializing
13:18:07 Listener report=SDS+HDR is active with 8 worker threads
13:18:07 Listener oltp=primary is active with 8 worker threads
13:18:07 Listener payroll=HDR+primary SLA report=SDS+HDR is active with 8 worker
 threads
13:18:07 Connection Manager successfully connected to ol_ids_1150_2
13:18:07 Arbitrator FOC string = ids_sds+ids_hdr,10
13:18:07 FOC[0] = ids_sds
13:18:07 FOC[1] = ids_hdr
13:18:07 FOC timeout = 10
13:18:07 Arbitrator is active on CM = cm_example
13:18:08 Connection Manager started successfully
```

It is also possible to specify all parameters on the command line without using a configuration file. For example:

```
oncmsm cm_example -s oltp=primary -s payroll=HDR+primary -s report=SDS+HDR
   -l cmlog -f ids_sds+ids_hdr,10
```

Establishing Connection Manager Redundancy and Failover

You can configure multiple instances of the Connection Manager on different machines in order to avoid having the Connection Manager itself become a single point of failure.

You can install and run the Connection Manager on an IDS server instance, or, to isolate the Connection Manager from database server failures, you can install it on a separate, non-IDS machine. In addition, you can configure the Connection Manager to run on a hardware platform other than the machines that make up the cluster. All that is required is the correct configuration of the sqlhosts file to indicate the connection host name and port number.

To further increase availability, you can configure multiple Connection Manager instances and use the failover feature of server groups within sqlhosts.

For example, the following section from the sqlhosts file creates a group named **oltp** that contains two Connection Manager instances named **oltp_cm1** and **oltp_cm2**:

```
oltp      group     -         -     e=oltp_cm2
oltp_cm1  onsoctcp  cm1_host  9298  g=oltp
oltp_cm2  onsoctcp  cm2_host  9298  g=oltp
```

By adding the primary server to the oltp group, it would be possible to maintain connectivity even if both Connection Manager instances failed.

For more information about creating and configuring groups, see "Client/Server Architecture" on page 3-1.

Monitoring the Connection Manager

Use **onstat -g cmsm** to display the various Connection Manager daemons that are attached to a server instance, and also the number of connections that the daemon has processed. See the *IBM Informix Administrator's Reference* for more information.

Example output from **onstat -g cmsm** with one instance of Connection Manager active:

```
 onstat -g cmsm

CM name  host      sla          define          foc            flag   connections
cm1      bia       oltp         primary         SDS+HDR+RSS,0     3             5
cm1      bia       report       (SDS+RSS)       SDS+HDR+RSS,0     3            16
```

Stopping the Connection Manager

You stop a Connection Manager instance when you no longer want connection redirection to occur.

To stop a Connection Manager instance use the **oncmsm** utility with the **-k** option.

```
oncmsm -k connection_manager_name
```

Modifying Connection Manager Options

You use the Connection Manager reload option to add or change service level agreements, failover parameters, log file name, or debug options.

Configuration options can be reloaded without stopping and restarting the Connection Manger. You can edit the existing configuration file, or overlay the original file with one that contains the new parameters. The new options take effect immediately.

To reload configuration options, use:

```
oncmsm -r connection_manager_name
```

The *connection_manager_name* is the name of the Connection Manager instance to reload. Because multiple instances of the Connection Manager can be active at the same time, it is necessary to specify the name of the Connection Manager instance.

For example, to reload the configuration file of a Connection Manager instance named cm_1:

1. Modify the existing Connection Manager configuration file with the new service level agreements, failover parameters, log file name, or debug option.
2. On the machine on which Connection Manager is running:

   ```
   oncmsm -r cm_1
   ```

   The Connection Manager loads the new parameters.

Use the onstat utility to determine the names of Connection Manager instances:

```
onstat -g cmsm
```

Failover with ISV Cluster Management Software

It is possible to use independent software vendor (ISV) cluster management software instead of the Connection Manager to manage failover processing in high-availability cluster environments. If the primary server in a high-availability cluster encounters a problem that requires a secondary server to assume the role of the primary, it is important that, prior to performing the actual failover, disk I/O is prohibited on the failed primary server and is allowed on the new primary server. In addition, network access to the failed primary server must be prevented. This is especially true for SD secondary servers, where disk corruption can occur if these steps are not done correctly.

The mechanism for enabling or disabling write operations to an SD secondary server in a high-availability cluster environment is known as *I/O Fencing*. *I/O Fencing* is configured using a callback script. When a failure of the primary server occurs, a script is executed on the secondary server that is to assume the role of the primary server. The script should call any I/O specific commands to enable or disable disk access. The script should enable write access to the shared disk on the server that is to become the primary server, and should disable write access to the shared disk on the failed server.

Use the FAILOVER_CALLBACK configuration parameter to specify the name of the script to run when a database server transitions from a secondary server to a primary server, or from a secondary server to a standard server. When configured,

the script specified by FAILOVER_CALLBACK is executed before the secondary server is switched to the primary server. See FAILOVER_CALLBACK in the *IBM Informix Administrator's Reference*.

If the script specified by FAILOVER_CALLBACK fails (that is, if it returns a non-zero exit code), the failover of the secondary to the primary (or standard) server will also fail. In this case, the DBA must manually perform the failover.

Chapter 24. Consistency Checking

In This Chapter

Informix database servers are designed to detect database server malfunctions or problems caused by hardware or operating-system errors. It detects problems by performing *assertions* in many of its critical functions. An *assertion* is a consistency check that verifies that the contents of a page, structure, or other entity match what would otherwise be assumed.

When one of these checks finds that the contents are not what they should be, the database server reports an *assertion failure* and writes text that describes the check that failed in the database server message log. The database server also collects further diagnostics information in a separate file that might be useful to IBM Informix Technical Support staff.

This chapter provides an overview of consistency-checking measures and ways of handling inconsistencies. It covers the following topics:

- Performing periodic consistency checking
- Dealing with data corruption
- Collecting advanced diagnostic information
- Monitoring the database server for disabling I/O errors

Performing Periodic Consistency Checking

To gain the maximum benefit from consistency checking and to ensure the integrity of dbspace backups, you should periodically take the following actions:

- Verify that all data and the database server overhead information is consistent.
- Check the message log for assertion failures while you verify consistency.
- Create a level-0 dbspace backup after you verify consistency.

The following sections describe each of these actions.

Verifying Consistency

Because of the time needed for this check and the possible contention that the check can cause, schedule this check for times when activity is at its lowest. You should perform this check just before you create a level-0 backup.

Run the commands shown in Table 24-1 as part of the consistency check.

Table 24-1. Checking Data Consistency

Type of Validation	Command
System catalog tables	**oncheck -cc**
Data	**oncheck -cD** *dbname*
Extents	**oncheck -ce**
Indexes	**oncheck -cI** *dbname*
Reserved pages	**oncheck -cr**
Logical logs and reserved pages	**oncheck -cR**

Table 24-1. Checking Data Consistency (continued)

Type of Validation	Command
Metadata and smart large objects	**oncheck -cs**

You can run each of these commands while the database server is in online mode. For information about how *each command* locks objects as it checks them and which users can perform validations, see **oncheck** in the *IBM Informix Administrator's Reference*.

In most cases, if one or more of these validation procedures detects an error, the solution is to restore the database from a dbspace backup. However, the source of the error might also be your hardware or operating system.

Validating System Catalog Tables

To validate system catalog tables, use the **oncheck -cc** command.

Each database contains its own system catalog, which contains information about the database tables, columns, indexes, views, constraints, stored procedures, and privileges.

If a warning appears when validation completes, its only purpose is to alert you that no records of a specific type were found. These warnings do not indicate any problem with your data, your system catalog, or even your database design. This warning indicates only that no synonym exists for any table; that is, the system catalog contains no records in the table **syssyntable**. For example, the following warning might appear if you validate system catalog tables for a database that has no synonyms defined for any table:

```
WARNING: No syssyntable records found.
```

However, if you receive an error message when you validate system catalog tables, the situation is quite different. Contact IBM Informix Technical Support immediately.

Validating Data Pages

To validate data pages, use the **oncheck -cD** command.

If data-page validation detects errors, try to unload the data from the specified table, drop the table, re-create the table, and reload the data. For information about loading and unloading data, see the *IBM Informix Migration Guide*. If this procedure does not succeed, perform a data restore from a storage-space backup.

Validating Extents

To validate extents in every database, use the **oncheck -ce** command.

Extents must not overlap. If this command detects errors, perform a data restore from a storage-space backup.

Validating Indexes

To validate indexes on each of the tables in the database, use the **oncheck -cI** command.

If this command detects errors, drop and re-create the affected index.

Validating Logical Logs

To validate logical logs and the reserved pages, use the **oncheck -cR** command.

Validating Reserved Pages

To validate reserved pages, use the **oncheck -cr** command.

Reserved pages are pages that reside at the beginning of the initial chunk of the root dbspace. These pages contain the primary database server overhead information. If this command detects errors, perform a data restore from storage-space backup.

This command might provide warnings. In most cases, these warnings call your attention to situations of which you are already aware.

Validating Metadata

Execute **oncheck -cs** for each database to validate metadata for all smart large objects in a database. If necessary, restore the data from a dbspace backup.

Monitoring for Data Inconsistency

If the consistency-checking code detects an inconsistency during database server operation, an assertion failure is reported to the database server message log. (See the message-log chapter in the *IBM Informix Administrator's Reference*.)

Reading Assertion Failures in the Message Log and Dump Files

Figure 24-1 shows the form that assertion failures take in the message log.

```
Assert Failed: Short description of what failed
   Who: Description of user/session/thread running at the time
   Result: State of the affected database server entity
   Action: What action the database server administrator should take
   See Also: file(s) containing additional diagnostics
```

Figure 24-1. Form of Assertion Failures in the Message Log

The `See Also:` line contains one or more of the following filenames:

- **af.xxx**
- **shmem.xxx**
- **gcore.xxx**
- **gcore.xxx**
- */path name*/**core**

In all cases, **xxx** is a hexadecimal number common to all files associated with the assertion failures of a single thread. The files **af.xxx**, **shmem.xxx**, and **gcore.xxx** are in the directory that the ONCONFIG parameter DUMPDIR specifies.

The file **af.xxx** contains a copy of the assertion-failure message that was sent to the message log, as well as the contents of the current, relevant structures and data buffers.

The file **shmem.xxx** contains a complete copy of the database server shared memory at the time of the assertion failure, but only if the ONCONFIG parameter DUMPSHMEM is set to 1 or to 2.

UNIX Only:

On UNIX, **gcore.xxx** contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPGCORE is set to 1 and your operating system supports the **gcore** utility. The **core** file contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPCORE is set to 1. The path name for the **core** file is the directory from which the database server was last invoked.

Validating Table and Tblspace Data

To validate table and tblspace data, use the **oncheck -cD** command on the database or table.

Most of the general assertion-failure messages are followed by additional information that usually includes the tblspace where the error was detected. If this check verifies the inconsistency, unload the data from the table, drop the table, re-create the table, and reload the data. Otherwise, no other action is needed.

In many cases, the database server stops immediately when an assertion fails. However, when failures appear to be specific to a table or smaller entity, the database server continues to run.

When an assertion fails because of inconsistencies on a data page that the database server accesses on behalf of a user, an error is also sent to the application process. The SQL error depends on the operation in progress. However, the ISAM error is almost always either -105 or -172, as follows:

```
-105 ISAM error: bad isam file format
-172 ISAM error: Unexpected internal error
```

For additional details about the objectives and contents of messages, see the chapter on message-log messages in the *IBM Informix Administrator's Reference*.

Retaining Consistent Level-0 Backups

After you perform the checks described in "Verifying Consistency" on page 24-1 without errors, create a level-0 backup. Retain this storage-space backup and all subsequent logical-log backup tapes until you complete the next consistency check. You should perform the consistency checks before every level-0 backup. If you do not, then at minimum keep all the tapes necessary to recover from the storage-space backup that was created immediately after the database server was verified to be consistent.

Dealing with Corruption

This section describes some of the symptoms of database server system corruption and actions that the database server or you, as administrator, can take to resolve the problems. Corruption in a database can occur as a consequence of hardware or operating-system problems, or from some unknown database server problems. Corruption can affect either data or database server overhead information.

Finding Symptoms of Corruption

The database server alerts the user and administrator to possible corruption in the following ways:

- Error messages reported to the application state that pages, tables, or databases cannot be found. One of the following errors is always returned to the application if an operation has failed because of an inconsistency in the underlying data or overhead information:

  ```
  -105 ISAM error: bad isam file format
  -172 ISAM error: Unexpected internal error
  ```

- Assertion-failure reports are written to the database server message log. They always indicate files that contain additional diagnostic information that can help you determine the source of the problem. See "Verifying Consistency" on page 24-1.
- The **oncheck** utility returns errors

Fixing Index Corruption

At the first indication of corruption, run the **oncheck -cI** command to determine if corruption exists in the index.

If you check indexes while the database server is in online mode, **oncheck** detects the corruption but does not prompt you for repairs. If corruption exists, you can drop and re-create the indexes using SQL statements while you are in online mode (the database server locks the table and index). If you run **oncheck -cI** in quiescent mode and corruption is detected, you are prompted to confirm whether the utility should attempt to repair the corruption.

Fixing I/O Errors on a Chunk

If an I/O error occurs during the database server operation, the status of the chunk on which the error occurred changes to *down*.

If a chunk is *down*, the **onstat -d** display shows the chunk status as `PD-` for a primary chunk and `MD-` for a mirror chunk. For an example of **onstat -d** output, see the *IBM Informix Administrator's Reference*.

In addition, the message log lists a message with the location of the error and a suggested solution. The listed solution is a possible fix, but does not always correct the problem.

If the down chunk is mirrored, the database server continues to operate using the mirror chunk. Use operating-system utilities to determine what is wrong with the down chunk and correct the problem. You must then direct the database server to restore mirror chunk data.

For information about recovering a mirror chunk, refer to "Recovering a Mirror Chunk" on page 18-6.

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates a stop action. Otherwise, the database server can continue to operate but cannot write to or read

from the down chunk or any other chunks in the dbspace of that chunk. You must take steps to determine why the I/O error occurred, correct the problem, and restore the dbspace from a backup.

If you take the database server to offline mode when a chunk is marked as down (D), you can restart the database server, provided that the chunk marked as down does not contain critical data (logical-log files, the physical log, or the root dbspace).

Collecting Diagnostic Information

Several ONCONFIG parameters affect the way in which the database server collects diagnostic information. Because an assertion failure is generally an indication of an unforeseen problem, notify IBM Informix Technical Support whenever one occurs. The diagnostic information collected is intended for the use of IBM Informix technical staff. The contents and use of **af**.*xxx* files and shared core are not further documented.

To determine the cause of the problem that triggered the assertion failure, it is critically important that you not destroy diagnostic information until IBM Informix Technical Support indicates that you can do so. The **af.xxx** file often contains information that they need to resolve the problem.

Several ONCONFIG parameters direct the database server to preserve diagnostic information whenever an assertion failure is detected or whenever the database server enters into an end sequence:

- DUMPDIR
- DUMPSHMEM
- DUMPCNT
- DUMPCORE
- DUMPGCORE

For more information about the configuration parameters, see the *IBM Informix Administrator's Reference*.

You decide whether to set these parameters. Diagnostic output can consume a large amount of disk space. (The exact content depends on the environment variables set and your operating system.) The elements of the output could include a copy of shared memory and a core dump.

Tip: A *core dump* is an image of a process in memory at the time that the assertion failed. On some systems, core dumps include a copy of shared memory. Core dumps are useful only if this is the case.

Database server administrators with disk-space constraints might prefer to write a script that detects the presence of diagnostic output in a specified directory and sends the output to tape. This approach preserves the diagnostic information and minimizes the amount of disk space used.

Disabling I/O Errors

Informix divides disabling I/O errors into two general categories: destructive and nondestructive. A disabling I/O error is destructive when the disk that contains a database becomes damaged in some way. This type of event threatens the integrity of data, and the database server marks the chunk and dbspace as down. The database server prohibits access to the damaged disk until you repair or replace the disk and perform a physical and logical restore.

A disabling I/O error is nondestructive when the error does not threaten the integrity of your data. Nondestructive errors occur when someone accidentally disconnects a cable, you somehow erase the symbolic link that you set up to point to a chunk, or a disk controller becomes damaged.

Before the database server considers an I/O error to be disabling, the error must meet two criteria. First, the error must occur when the database server attempts to perform an operation on a chunk that has at least one of the following characteristics:

- The chunk has no mirror.
- The primary or mirror companion of the chunk under question is offline.

Second, the error must occur when the database server attempts unsuccessfully to perform one of the following operations:

- Seek, read, or write on a chunk
- Open a chunk
- Verify that chunk information on the first used page is valid

  The database server performs this verification as a sanity check immediately after it opens a chunk.

You can prevent the database server from marking a dbspace as down while you investigate disabling I/O errors. If you find that the problem is trivial, such as a loose cable, you can bring the database server offline and then online again without restoring the affected dbspace from backup. If you find that the problem is more serious, such as a damaged disk, you can use **onmode -O** to mark the affected dbspace as down and continue processing.

Monitoring the Database Server for Disabling I/O Errors

The database server notifies you about disabling I/O errors in two ways:

- Message log
- Event alarms

Using the Message Log to Monitor Disabling I/O Errors

The database server sends the following message to the message log when a disabling I/O error occurs:

```
Assert Failed: Chunk {chunk-number} is being taken OFFLINE.
Who: Description of user/session/thread running at the time
Result: State of the affected database server entity
Action: What action the database server administrator should take
See Also: DUMPDIR/af.uniqid containing more diagnostics
```

The result and action depend on the current setting of ONDBSPACEDOWN, as described in the following table.

ONDBSPACEDOWN Setting	Result	Action
0	Dbspace {*space_name*} is disabled.	Restore dbspace {*space_name*}.
	Blobspace {*space_name*} is disabled.	Restore blobspace {*space_name*}.
1	The database server must stop.	Shut down and restart the database server.
2	The database server blocks at next checkpoint.	Use **onmode -k** to shut down, or use **onmode -O** to override.

The value of ONDBSPACEDOWN has no affect on temporary dbspaces. For temporary dbspaces, the database server continues processing regardless of the ONDBSPACEDOWN setting. If a temporary dbspace requires fixing, you can drop and recreate it.

For more information about interpreting messages that the database server sends to the message log, see the chapter about message-log messages in the *IBM Informix Administrator's Reference*.

Using Event Alarms to Monitor Disabling I/O Errors

When a dbspace incurs a disabling I/O error, the database server passes the following values as parameters to your event-alarm executable file.

Parameter
Value

Severity
4 (Emergency)

Class 5

Class message
Dbspace is disabled: 'dbspace-name'

Specific message
Chunk {chunk-number} is being taken OFFLINE.

If you want the database server to use event alarms to notify you about disabling I/O errors, write a script that the database server executes when it detects a disabling I/O error. For information about how to set up this executable file that you write, see the appendix on event alarms and the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

Using No Bad-Sector Mapping

Dynamic Server relies on the operating system of your host computer for bad-sector mapping. The database server learns of a bad sector or a bad track when it receives a failure return code from a system call. When this situation occurs, the database server retries the access several times to ensure that the condition is not spurious. If the condition is confirmed, the database server marks as *down* the chunk where the read or write was attempted.

The database server cannot take any action to identify the bad cylinder, track, or sector location because the only information available is the byte displacement within the chunk where the I/O operation was attempted.

If the database server detects an I/O error on a chunk that is *not* mirrored, it marks the chunk as down. If the down chunk contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates a stop action. Otherwise, the database server can continue to operate, but applications cannot access the down chunk until its dbspace is restored.

Part 5. Distributed Data

Chapter 25. Multiphase Commit Protocols

In This Chapter

A *two-phase commit protocol* ensures that transactions are uniformly committed or rolled back across multiple database servers. You can use Informix database servers with IBM Informix Enterprise Gateway products or transaction managers to manipulate data in non-Informix databases. Distributed queries across Informix database servers support two-phase commit.

The *heterogeneous commit protocol* ensures that updates to one or more Informix databases and *one* non-Informix database in a single transaction are uniformly committed or rolled back.

This chapter contains information on the use of the two-phase commit protocol. For information on recovering manually from a failed two-phase commit transaction, see Chapter 26, "Recovering Manually from Failed Two-Phase Commit," on page 26-1.

This chapter also contains information on using transaction support for XA-compliant, external data sources, which can participate in two-phase commit transactions. See "Using Dynamic Server Transaction Support for XA-Compliant, External Data Sources" on page 25-2.

Transaction Managers

Transaction managers support two-phase commit and roll back. For example, if your database is Informix, your accounting system is Oracle, and your remittance system is Sybase, you can use a transaction manager to communicate between the different databases. You also can use transaction managers to ensure data consistency between Informix or non-Informix databases by using distributed transactions instead of Enterprise Replication or High-Availability Data Replication.

Using the TP/XA Library With a Transaction Manager

A *global transaction* is a distributed query where more than one database server is involved in the query. A global transaction environment has the following parts:

- The client application
- The resource manager (Informix database server)
- The transaction manager (third-party software)

TP/XA is a library of functions that lets the database server act as a resource manager in the X/Open DTP environment. Install the TP/XA library as part of IBM Informix ESQL/C to enable communication between a third-party transaction manager and the database server. The X/Open environment supports large-scale, high-performance OLTP applications. For more information, see the *IBM Informix TP/XA Programmer's Manual*.

Use TP/XA when your database has the following characteristics:

- Data is distributed across multivendor databases
- Transactions include Informix and non-Informix data

Using Microsoft Transaction Server (MTS/XA)

The database server supports the Microsoft® Transaction Server (MTS/XA) as a transaction manager in the XA environment. To use MTS/XA, install IBM Informix Client Software Development Kit, the latest version of IBM Informix ODBC Driver, and MTS/XA. MTS/XA works on Windows. For more information, contact IBM Informix Technical Support, and see the *IBM Informix Client Products Installation Guide* and the MTS/XA documentation.

Using Dynamic Server Transaction Support for XA-Compliant, External Data Sources

The Dynamic Server Transaction Manager, which is an integral part of Dynamic Server, not a separate module, recognizes XA-compliant, external data sources. These data sources can participate in two-phase commit transactions.

The transaction manager invokes support routines for each XA-compliant, external data source that participates in a distributed transaction at a particular transactional event, such as prepare, commit, or rollback. This interaction conforms to X/Open XA interface standards.

Transaction support for XA-compliant, external data sources, which are also called *resource managers*, enables you to:

- Create XA-compliant, external data source types and instances of XA-compliant, external data sources.
- Create or modify a user-defined routine (UDR), virtual table interface, or virtual index interface to enable XA-compliant data sources to provide data access mechanisms for external data from XA-compliant data sources.

  The MQ DataBlade Module is an example of a set of UDRs that provide this type of external data access.
- Register XA-compliant, external data sources with Dynamic Server.
- Unregister XA-compliant, external data sources.
- Use multiple XA-compliant, external data sources within the same global transaction.

The transaction coordination with an XA-compliant, external data source is supported only in Informix logged databases and ANSI-compliant databases, since these databases support transactions. Transaction coordination with an XA-compliant, external data source is not supported in non-logged databases.

You can use the following DDL statements, which are extensions to SQL statements to manage XA data source types and data sources:

Statement	Description
CREATE XADATASOURCE TYPE	Creates a type of XA-compliant, external data source
CREATE XADATASOURCE	Creates an instance of an XA-compliant, external data source
DROP XADATASOURCE	Deletes an instance of an XA-compliant, external data source
DROP XADATASOURCE TYPE	Deletes a type of XA-compliant, external data source

For more information on these statements, see the *IBM Informix Guide to SQL: Syntax*.

The interaction between Dynamic Server and an XA-compliant, external data source occurs through a set of user-defined XA-support routines, such as **xa_open**, **xa_end**, **xa_commit**, and **xa_prepare**. You create these support routines before using the CREATE XADATASOURCE TYPE statement. For more information, see the *IBM Informix DataBlade API Programmer's Guide*.

After you create an external XA-compliant data source, you can register the data source to a current transaction and you can unregister the data source using the **mi_xa_register_xadatasource()** or **ax_reg()** and **mi_xa_unregister_xadatasource()** or **ax_unreg()** functions. In a distributed environment, you must register a data source at the local, coordinator server. Registration is transient, lasting only for the duration of the transaction. For more information on using these functions, see the *IBM Informix DataBlade API Function Reference* and the *IBM Informix DataBlade API Programmer's Guide*.

Use the following **onstat** options to display information about transactions involving XA-compliant data sources:

onstat Option	**What XA-Compliant Data Source Information This Command Displays**
onstat -x	Displays information on XA participants in a transaction.
onstat -G	Displays information on XA participants in a global transaction.
onstat -g ses *session id*	Displays session information, including information about XA data sources participating in a transaction.

The IBM Informix MQ DataBlade provides external data access mechanisms for XA data sources. For information on this DataBlade Module, see the *IBM Informix Database Extensions User's Guide*.

Using Loosely-Coupled and Tightly-Coupled Modes

The database server supports XA global transactions in loosely-coupled and tightly-coupled modes:

- *Loosely coupled mode* means that the different database servers coordinate transactions, but do not share resources. The records from all branches of the transactions display as separate transactions in the logical log.
- *Tightly coupled mode* means that the different database servers coordinate transactions and share resources such as locking and logging. The records from all branches of the transactions display as a single transaction in the logical log.

The Tuxedo Transaction Manager, provided by BEA systems, supports loosely-coupled mode. Tuxedo operates on both UNIX and Windows.

Windows Only:

The MTS/XA Transaction Manager, which operates only on Windows, supports the tightly-coupled mode. MTS tightly-coupled transaction support on the database server includes:

- Support for application programs with two tiers (a business-logic layer and a data-access layer).
- Connection pooling and session pooling.

MTS tightly-coupled transaction support does not affect existing loosely-coupled-transaction support. The same database server can use both loosely-coupled and tightly-coupled transaction support at the same time.

MTS tightly-coupled transaction support has the following restrictions:

- Temporary tables are limited to one transaction branch. Different transaction branches within one global transaction cannot share a temporary table.
- Different transaction branches within one global transaction cannot share cursors.
- Different transaction branches within one global transaction cannot share an isolation level or lock-wait mode. The isolation level and lock-wait mode of each transaction branch must be set individually or set to the default level. If you want the same isolation level for all transaction branches, you must use SQL to specify this information for each transaction branch.

For a complete list of supported transaction managers, contact your marketing representative.

Two-Phase Commit Protocol

The two-phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction. The two-phase commit protocol ensures that all participating database servers receive and implement the same action (either to commit or to roll back a transaction), regardless of local or network failure.

If any database server is unable to commit its portion of the transaction, *all* database servers participating in the transaction must be prevented from committing their work.

When the Two-Phase Commit Protocol Is Used

A database server automatically uses the two-phase commit protocol for any transaction that modifies data on multiple database servers.

For example, suppose three database servers that have the names **australia**, **italy**, and **france**, are connected, as shown in Figure 25-1.

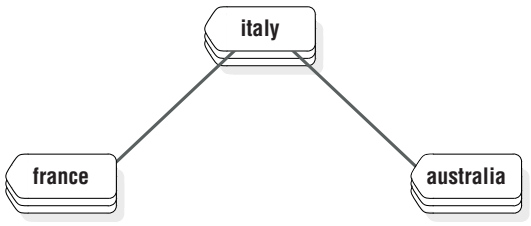


Figure 25-1. Connected Database Servers

If you execute the commands shown in Figure 25-2 on page 25-5, the result is one update and two inserts at three different database servers.

```
CONNECT TO stores_demo@italy
BEGIN WORK
   UPDATE stores_demo:manufact SET manu_code = 'SHM' WHERE manu_name = 'Shimara'
   INSERT INTO stores_demo@france:manufact VALUES ('SHM', 'Shimara', '30')
   INSERT INTO stores_demo@australia:manufact VALUES ('SHM', 'Shimara', '30')
COMMIT WORK
```

Figure 25-2. Example of a Distributed Transaction

Two-Phase Commit Concepts

Every global transaction has a *coordinator* and one or more *participants*, defined as follows:

- The *coordinator* directs the resolution of the global transaction. It decides whether the global transaction should be committed or stopped.

  The two-phase commit protocol always assigns the role of coordinator to the *current* database server. The role of coordinator cannot change during a single transaction. In the sample transaction in Figure 25-2, the coordinator is **italy**. If you change the first line in this example to the following statement, the two-phase commit protocol assigns the role of coordinator to **france**:

  ```
  CONNECT TO stores_demo@france
  ```

  Use the **onstat -x** option to display the coordinator for a distributed transaction. For more information, see "Monitoring a Global Transaction" on page 25-14.
- Each *participant* directs the execution of one *transaction branch*, which is the part of the global transaction involving a single local database. A global transaction includes several transaction branches when:
  - An application uses multiple processes to work for a global transaction
  - Multiple remote applications work for the same global transaction

  In Figure 25-1 on page 25-4, the participants are **france** and **australia**. The coordinator database server, **italy**, also functions as a participant because it is also doing an update.

The two-phase commit protocol relies on two kinds of communication, *messages* and *logical-log records*:

- Messages pass between the coordinator and each participant. Messages from the coordinator include a transaction identification number and instructions (such as `prepare to commit`, `commit`, or `roll back`). Messages from each participant include the transaction status and reports of action taken (such as `can commit` or `cannot commit`, `committed`, or `rolled back`).
- Logical-log records of the transaction are kept on disk or tape to ensure data integrity and consistency, even if a failure occurs at a participating database server (participant or coordinator).

  For more details, refer to "Two-Phase Commit and Logical-Log Records" on page 25-15.

Phases of the Two-Phase Commit Protocol

In a two-phase commit transaction, the coordinator sends all the data modification instructions (for example, inserts) to all the participants. Then, the coordinator starts the two-phase commit protocol. The two-phase commit protocol has two parts, the *precommit phase* and the *postdecision phase*.

Precommit Phase

During the precommit phase, the coordinator and participants perform the following dialog:

1. **Coordinator**. The coordinator directs each participant database server to prepare to commit the transaction.
2. **Participants**. Every participant notifies the coordinator whether it can commit its transaction branch.
3. **Coordinator**. The coordinator, based on the response from each participant, decides whether to commit or roll back the transaction. It decides to commit only if *all* participants indicate that they can commit their transaction branches. If any participant indicates that it is *not* ready to commit its transaction branch (or if it does not respond), the coordinator decides to end the global transaction.

Postdecision Phase

During the postdecision phase, the coordinator and participants perform the following dialog:

1. **Coordinator**: The coordinator writes the commit record or rollback record to the coordinator's logical log and then directs each participant database server to either commit or roll back the transaction.
2. **Participants**: If the coordinator issued a commit message, the participants commit the transaction by writing the commit record to the logical log and then sending a message to the coordinator acknowledging that the transaction was committed. If the coordinator issued a rollback message, the participants roll back the transaction but do not send an acknowledgment to the coordinator.
3. **Coordinator**: If the coordinator issued a message to commit the transaction, it waits to receive acknowledgment from each participant before it ends the global transaction. If the coordinator issued a message to roll back the transaction, it does not wait for acknowledgments from the participants.

How the Two-Phase Commit Protocol Handles Failures

The two-phase commit protocol is designed to handle system and media failures in such a way that data integrity is preserved across all the participating database servers. The two-phase commit protocol performs an *automatic recovery* if a failure occurs.

Types of Failures That Automatic Recovery Handles

The following events can cause the coordinating thread or the participant thread to terminate or hang, thereby requiring automatic recovery:

- System failure of the coordinator
- System failure of a participant
- Network failure
- Termination of the coordinating thread by the administrator
- Termination of the participant thread by the administrator

Administrator's Role in Automatic Recovery

The only role of the administrator in automatic recovery is to bring the coordinator or participant (or both) back online after a system or network failure.

Important: A slow network cannot, and should not, trigger automatic recovery. None of the recovery mechanisms described here go into effect unless a coordinator system fails, a network fails, or the administrator terminates the coordinating thread.

Automatic-Recovery Mechanisms for Coordinator Failure

If the coordinating thread fails, each participant database server must decide whether to initiate automatic recovery *before* it commits or rolls back the transaction or *after* it rolls back a transaction. This responsibility is part of the presumed-end optimization. (See "Presumed-End Optimization.")

Automatic-Recovery Mechanisms for Participant Failure

Participant recovery occurs whenever a participant thread precommits an item of work that is terminated before the two-phase commit protocol can be completed. The goal of participant recovery is to complete the two-phase commit protocol according to the decision reached by the coordinator.

Participant recovery is driven by either the coordinator or the participant, depending on whether the coordinator decided to commit or to roll back the global transaction.

Important: To support automatic recovery after a subordinate server is shut down or restarted while a cross-server transaction is open, the **sqlhosts** file should include an entry for every database server from which distributed operations might be initiated. During automatic recovery, the name of the coordinator is recovered from the logical logs, and the subordinate server reconnects with the coordinator to complete the transaction. Because the coordinator always identifies itself to the participants using the name that is in the DBSERVERNAME configuration parameter in its own ONCONFIG file, the DBSERVERNAME of the coordinator should be an Internet protocol connection name known to the participants. The subordinate server should be able to connect to the coordinator using the DBSERVERNAME of the coordinator.

Presumed-End Optimization

Presumed-end optimization is a term that describes how the two-phase commit protocol handles the rollback of a transaction.

Rollback is handled in the following manner. When the coordinator determines that the transaction must be rolled back, it sends a message to all the participants to roll back their piece of work. The coordinator does not wait for an acknowledgment of this message, but proceeds to close the transaction and remove it from shared memory. If a participant tries to determine the status of this transaction—that is, find out whether the transaction was committed or rolled back (during participant recovery, for example)—it does not find any transaction status in shared memory. The participant must interpret this as meaning that the transaction was rolled back.

Independent Actions

An independent action in the context of two-phase commit is an action that occurs independently of the two-phase commit protocol. Independent actions might or might not be in opposition to the actions that the two-phase commit protocol specifies. If the action *is* in opposition to the two-phase commit protocol, the action results in an error or a *heuristic decision*. Heuristic decisions can result in an inconsistent database and require manual two-phase commit recovery. Manual recovery is an extremely complicated administrative procedure that you should try to avoid. (For a discussion of the manual-recovery process, see Chapter 26, "Recovering Manually from Failed Two-Phase Commit," on page 26-1.)

Situations That Initiate Independent Action

Independent action during a two-phase commit protocol is rare, but it can occur in the following situations:

- The participant's piece of work develops into a long-transaction error and is rolled back.
- An administrator stops a participant thread during the postdecision phase of the protocol with **onmode -z**.
- An administrator ends a participant transaction (piece of work) during the postdecision phase of the protocol with **onmode -Z**.
- An administrator ends a global transaction at the coordinator database server with **onmode -z** or **onmode -Z** *after* the coordinator issued a commit decision *and* became aware of a participant failure. This action always results in an error, specifically error -716.

Possible Results of Independent Action

As mentioned earlier, not all independent actions are in opposition to the two-phase commit protocol. Independent actions can yield the following three possible results:

- Successful completion of the two-phase commit protocol
- An error condition
- A heuristic decision

If the action is not in opposition to the two-phase protocol, the transaction should either commit or roll back normally. If the action ends the global transaction prematurely, an error condition results. Ending the global transaction at the coordinator is not considered a heuristic decision. If the action is in opposition to the two-phase commit protocol, a heuristic decision results. All these situations are discussed in the sections that follow.

Independent Actions That Allow Transactions to Complete Successfully

Independent actions are not necessarily in opposition to the two-phase commit protocol. For example, if a piece of work at a participant database server is rolled back because it developed into a long transaction, *and* the coordinator issues a decision to roll back the global transaction, the database remains consistent.

Independent Actions That Result in an Error Condition

If you, as administrator at the coordinator database server, execute either **onmode -z** (stop the coordinator thread) or **onmode -Z** (stop the global transaction) after the coordinator issues its final *commit* decision, you are removing all knowledge of the transaction from shared memory at the coordinator database server.

This action is not considered a heuristic decision because it does not interfere with the two-phase protocol; it is either acceptable, or it interferes with participant recovery and causes an error.

The action is acceptable any time that all participants are able to commit the transaction without difficulty. In this case, your action to end the transaction forcibly is superfluous. The indication that you executed **onmode -Z** reaches the coordinator only when the coordinator is preparing to terminate the transaction.

In practice, however, you would probably consider executing **onmode -z** or **onmode -Z** at the coordinator database server only if you were attempting to hasten the conclusion of a global transaction that has remained open for an unusually long period. In this scenario, the source of the problem is probably a failure at some participant database server. The coordinator has not received acknowledgment that the participant committed its piece of work, and the coordinator is attempting to establish communication with the participant to investigate.

If you execute either **onmode -z** or **onmode -Z** while the coordinator is actively trying to reestablish communication, the coordinating thread obeys your instruction to die, but not before it writes error -716 into the database server message log. The action is considered an error because the two-phase commit protocol was forcibly broken, preventing the coordinator from determining whether the database is consistent.

Stopping a global transaction at a coordinator database server is not considered a heuristic decision, but it can result in an inconsistent database. For example, if the participant eventually comes back online and does not find the global transaction in the coordinator shared memory, it rolls back its piece of work, thereby causing a database inconsistency.

Independent Actions That Result in Heuristic Decisions

Some independent actions can develop into heuristic decisions when *both* of the following conditions are true:

- The participant database server already sent a `can commit` message to the coordinator and then rolls back.
- The coordinator's decision is to commit the transaction.

When both conditions are true, the net result is a global transaction that is inconsistently implemented (committed by one or more database servers and rolled back by another). The database becomes inconsistent.

The following two heuristic decisions are possible:

- Heuristic rollback (described in "The Heuristic Rollback Scenario" on page 25-10)
- Heuristic end transaction (described in "The Heuristic End-Transaction Scenario" on page 25-12)

After a heuristic rollback or end transaction occurs, you might have to perform manual recovery, a complex and time-consuming process. You need to understand heuristic decisions fully in order to avoid them. Always be wary of executing **onmode -z** or **onmode -Z** within the context of two-phase commit.

The Heuristic Rollback Scenario

In a *heuristic rollback*, either the database server or the administrator rolls back a piece of work that has already sent a `can commit` message.

Conditions That Result in a Heuristic Rollback

The following two conditions can cause a heuristic rollback:

- The logical log fills to the point defined by the LTXEHWM configuration parameter. (See the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.) The source of the long-transaction condition is a piece of work being performed on behalf of a global transaction.
- An administrator executes **onmode -z *session_id*** to stop a database server thread that is executing a piece of work being performed on behalf of a global transaction.

In either case, if the piece of work has already sent a `can commit` message to its coordinator, the action is considered a heuristic decision.

Condition 1: Logical Log Fills to a High-Watermark:

Under two-phase commit, a participant database server that is waiting for instructions from the coordinator is blocked from completing its transaction. Because the transaction remains open, the logical-log files that contain records associated with this transaction cannot be freed. The result is that the logical log continues to fill because of the activity of concurrent users.

If the logical log fills to the value of the long-transaction high-watermark (LTXHWM) while the participant is waiting, the database server directs all database server threads that own long transactions to begin rolling them back. If a piece of work that is precommitted is the offending long transaction, the database server has initiated a heuristic rollback. That is, this database server is rolling back a precommitted piece of work without the instruction or knowledge of the coordinator.

Under two-phase commit, the logical-log files that contain records associated with the piece of work are considered open until an ENDTRANS logical-log record is written. This type of transaction differs from a transaction involving a single database server where a rollback actually closes the transaction.

The logical log might continue to fill until the exclusive high-watermark is reached (LTXEHWM). If this happens, all user threads are suspended except those that are currently rolling back or currently committing. In the two-phase commit scenario, the open transaction prevents you from backing up the logical-log files and freeing space in the logical log. Under these specific circumstances, the logical log can fill completely. If this happens, the participant database server shuts down, and you must perform a data restore.

Condition 2: System Administrator Executes onmode -z:

You, as administrator, can decide to initiate a heuristic rollback of a precommitted piece of work by executing **onmode -z**. You might make this decision because you want to free the resources that are held by the piece of work. (If you stop the participant thread by executing **onmode -z**, you free all locks and shared-memory resources that are held by the participant thread even though you do not end the transaction.)

Results of a Heuristic Rollback

This section describes what happens at both the coordinator and participant when a heuristic rollback occurs and how this process can result in an inconsistent database:

1. At the participant database server where the rollback occurred, a record is placed in the database server logical log (type HEURTX). Locks and resources held by the transaction are freed. The participant thread writes the following message in the database server message log, indicating that a long-transaction condition and rollback occurred:

   ```
   Transaction Completed Abnormally (rollback):
   tx=address flags=0xnn
   ```

2. The coordinator issues postdecision phase instructions to commit the transaction.

   The participant thread at the database server where the heuristic rollback occurred returns error message -699 to the coordinator as follows:

   ```
   -699 Transaction heuristically rolled back.
   ```

   This error message is not returned to the application at this point; it is an internal notification to the coordinator. The coordinator waits until all participants respond to the commit instruction. The coordinator does not determine database consistency until all participants report.

3. The next steps depend on the actions that occur at the other participants. Two possible situations are possible.

Situation 1: Coordinator Issues a Commit and All Participants Report Heuristic Rollbacks:

The coordinator gathers all responses from participants. If every participant reports a heuristic rollback, the following events occur as a consequence:

1. The coordinator writes the following message to its own database- server message log:

   ```
   Transaction heuristically rolled back.
   ```

2. The coordinator sends a message to all participants to end the transaction.
3. Each participant writes an ENDTRANS record in its logical-log buffer. (The transaction entry is removed from the transaction table.)
4. The coordinator returns error -699 to the application, as follows:

   ```
   -699 Transaction heuristically rolled back.
   ```

5. In this situation, all databases remain consistent.

Situation 2: Coordinator Issued a Commit; One Participant Commits and One Reports a Heuristic Rollback:

The coordinator gathers all responses from participants. If at least one participant reports a heuristic rollback and at least one reports an acknowledgment of a commit, the result is referred to as a *mixed-transaction result*. The following events occur as a consequence:

1. The coordinator writes the following message to its own database server message log:

   ```
   Mixed transaction result. (pid=nn user=userid)
   ```

   The pid value is the user-process identification number of the coordinator process. The user value is the user ID associated with the coordinator process. Associated with this message are additional messages that list each of the participant database servers that reported a heuristic rollback. The additional messages take the following form:

   ```
   Participant database server dbservername heuristically rolled back.
   ```

2. The coordinator sends a message to each participant that heuristically rolled back its piece of work, directing each one to end the transaction.
3. Each participant writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the transaction table.)
4. The coordinator writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)
5. The coordinator returns error -698 to the application, as follows:

   ```
   -698 Inconsistent transaction. Number and names of servers rolled back.
   ```

6. Associated with this error message is the list of participant database servers that reported a heuristic rollback. If a large number of database servers rolled back the transaction, this list could be truncated. The complete list is always included in the message log for the coordinator database server.

In this situation, examine the logical log at each participant database server site and determine whether your database system is consistent. (See "Determining If a Transaction Was Implemented Inconsistently" on page 26-1.)

The Heuristic End-Transaction Scenario

A *heuristic end transaction* is an independent action taken by the administrator to roll back a piece of work and remove all information about the transaction from the transaction table. The heuristic end-transaction process is initiated when the administrator executes the **onmode -Z *address*** command.

Whenever you initiate a heuristic end transaction by executing **onmode -Z**, you remove critical information required by the database server to support the two-phase commit protocol and its automatic-recovery features. If you execute **onmode -Z**, it becomes your responsibility to determine whether your networked database system is consistent.

When to Perform a Heuristic End Transaction

You should execute the **onmode -Z** option to initiate a heuristic end transaction in only one, rare, situation. This situation occurs when a piece of work that has been heuristically rolled back remains open, preventing your logical-log files from becoming free. As a result, the logical log is dangerously close to full.

In general, the coordinator issues its commit-or-rollback decision within a reasonable period of time. However, if the coordinator fails and does not return online to end a transaction that was heuristically rolled back at your participant database server, you might face a serious problem.

The problem scenario begins in this way:

1. The participant thread that is executing a piece of work on behalf of a global transaction has sent a `can commit` response to the coordinator.

2. The piece of work waits for instructions from the coordinator.
3. While the piece of work is waiting, the logical log fills past the long-transaction high-watermark.
4. The piece of work that is waiting for instructions is the source of the long transaction. The participant database server directs the executing thread to roll back the piece of work. This action is a heuristic rollback.
5. The participant continues to wait for the coordinator to direct it to end the transaction. The transaction remains open. The logical log continues to fill.

If the coordinator contacts the participant and directs it to end the transaction in a reasonable period of time, no problem develops. The serious problem arises if the heuristic rollback occurs at a participant database server and subsequently the coordinator fails, preventing the coordinator from directing the participant to end the transaction.

As a consequence, the transaction remains open. The open transaction prevents you from backing up logical-log files and freeing space in the logical log. As the logical log continues to fill, it might reach the point specified by the exclusive-access, long-transaction high-watermark (LTXEHWM). If this point is reached, normal processing is suspended. At some point after the high-watermark is reached, you must decide if the open transaction is endangering your logical log. The danger is that if the logical log fills completely, the database server shuts down, and you must perform a data restore.

You must decide whether to end the transaction and protect your system against the possibility of filling the logical log, despite all the problems associated with executing **onmode -Z**, or to wait and see if communication with the coordinator can be reestablished in time to end the transaction before the logical log fills.

How to Use onmode -Z

The **onmode -Z *address*** command is intended for use only if communication between the coordinator and the participant is broken. To ensure that communication is really broken, the **onmode -Z** command does not execute unless the thread that was executing the piece of work has been dead for the amount of time specified by TXTIMEOUT. For more information on this option, see the *IBM Informix Administrator's Reference*.

The *address* parameter is obtained from **onstat -x output**. For more information on the **onstat -x** option, see the *IBM Informix Administrator's Reference*.

Action When the Transaction Is Ended Heuristically

When you execute **onmode -Z**, you direct the **onmode** utility to remove the participant transaction entry, which is located at the specified address, from the transaction table.

Two records are written in the logical log to document the action. The records are type ROLLBACK and ENDTRANS, or if the transaction was already heuristically rolled back, ENDTRANS only. The following message is written to the participant database server message log:

```
(time_stamp) Transaction Completed Abnormally (endtx): tx=address
flags:0xnn user username tty ttyid
```

The coordinator receives an error message from the participant where the **onmode -Z** occurred, in response to its COMMIT instruction. The coordinator queries the participant database server, which no longer has information about the transaction. The lack of a transaction-table entry at the participant database server indicates that the transaction committed. The coordinator assumes that the acknowledgment message was sent from the participant, but somehow it was not received. Because the coordinator *does not know* that this participant's piece of work did not commit, it does not generate messages indicating that the global transaction was inconsistently implemented. Only the administrator who executed the **onmode -Z** command is aware of the inconsistent implementation.

Monitoring a Global Transaction

Use the **onstat -x** command to track open transactions and determine whether they have been heuristically rolled back.

For example, in the output, an `H` flag in the **flags** field identifies a heuristic rollback, the `G` flag identifies a global transaction, the `L` flag indicates loosely-coupled mode, and the `T` flag indicates tightly-coupled mode.

The **curlog** and **logposit** fields provide the exact position of a logical-log record. If a transaction is not rolling back, **curlog** and **logposit** describe the position of the most recently written log record. When a transaction is rolling back, these fields describe the position of the most recently "undone" log record. As the transaction rolls back, the **curlog** and **logposit** values decrease. In a long transaction, the rate at which the **logposit** and **beginlg** values converge can help you estimate how much longer the rollback is going to take.

For more information on and an example of **onstat -x** output, see the *IBM Informix Administrator's Reference*.

You also can use the **onstat -u** and **onstat -k** commands to track transactions and the locks that they hold. For details, see the monitoring transactions section in your *IBM Informix Performance Guide*. For a description of the fields that **onstat -x** displays, see the *IBM Informix Administrator's Reference*.

Two-Phase Commit Protocol Errors

The following two-phase commit protocol errors require special attention from the administrator.

Error Number
Description

-698 If you receive error -698, a heuristic rollback has occurred and has caused an inconsistently implemented transaction. The circumstances leading up to this event are described in "Results of a Heuristic Rollback" on page 25-11. For an explanation of how the inconsistent transaction developed and to learn the options available to you, refer to this discussion.

-699 If you receive error -699, a heuristic rollback has occurred. The circumstances leading up to this event are described in "Results of a Heuristic Rollback" on page 25-11. For an explanation of how the inconsistent transaction developed, refer to this discussion.

-716 If you receive error -716, the coordinating thread has been terminated by

administrator action after it issued its final decision. This scenario is described under "Independent Actions That Result in an Error Condition" on page 25-9.

Two-Phase Commit and Logical-Log Records

The database server uses logical-log records to implement the two-phase commit protocol. You can use these logical-log records to detect heuristic decisions and, if necessary, to help you perform a manual recovery. (See Chapter 26, "Recovering Manually from Failed Two-Phase Commit," on page 26-1.)

The following logical-log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

For information about these logical-log records, see the chapter on interpreting the logical log in the *IBM Informix Administrator's Reference*.

This section examines the sequence of logical-log records that are written during the following database server scenarios:

- A transaction is committed.
- A piece of work is heuristically rolled back.
- A piece of work is heuristically ended.

Logical-Log Records When the Transaction Commits

Figure 25-3 on page 25-16 illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.

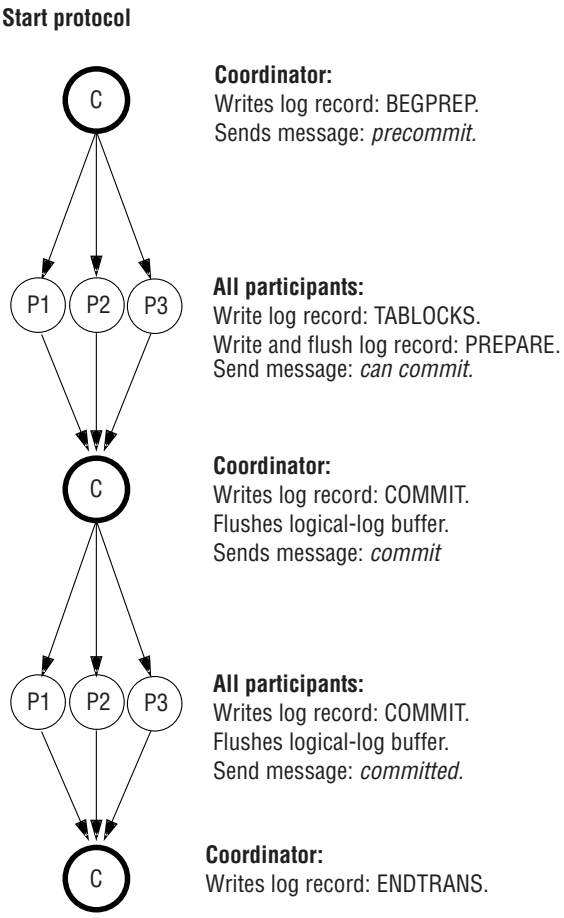


Figure 25-3. Logical-Log Records Written During a Committed Transaction

Some of the logical-log records must be flushed from the logical-log buffer immediately; for others, flushing is not critical.

The coordinator's commit-work record (COMMIT record) contains all information needed to initiate the two-phase commit protocol. It also serves as the starting point for automatic recovery in the event of a failure on the coordinator's host computer. Because this record is critical to recovery, it is not allowed to remain in the logical-log buffer. The coordinator must immediately flush the COMMIT logical-log record.

The participants in Figure 25-3 must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. As part of recovery, the participant might query the coordinator to learn the final disposition of this transaction.

Flushing the participant's COMMIT record ensures that, if the participant's host computer fails, the participant has a record of what action it took regarding the transaction. To understand why this information is crucial, consider the situation in which a participant crashes after the PREPARE record is written but before the COMMIT record flushes. After fast recovery, the PREPARE record is restored, but the COMMIT record is lost (because it was in the logical-log buffer at the time of

the failure). The existence of the PREPARE record would initiate a query to the coordinator about the transaction. However, the coordinator would know nothing of the transaction, because it ended the transaction after it received the participant's acknowledgment that the commit occurred. In this situation, the participant would interpret the lack of information as a final direction to roll back the transaction. The two-phase commit protocol requires the participant's COMMIT record to be flushed immediately to prevent this kind of misunderstanding.

Logical-Log Records Written During a Heuristic Rollback

Figure 25-4 on page 25-18 illustrates the sequence in which the database server writes the logical-log records during a heuristic rollback. Because a heuristic rollback only occurs after the participant sends a message that it can commit and the coordinator sends a message to commit, the first phase of this protocol is the same as that shown in Figure 25-3 on page 25-16. When a heuristic rollback occurs, the rollback is assumed to be the consequence of a long-transaction condition that occurs at the Participant 1 (P1) database server. The end result is a transaction that is inconsistently implemented. See "The Heuristic Rollback Scenario" on page 25-10.

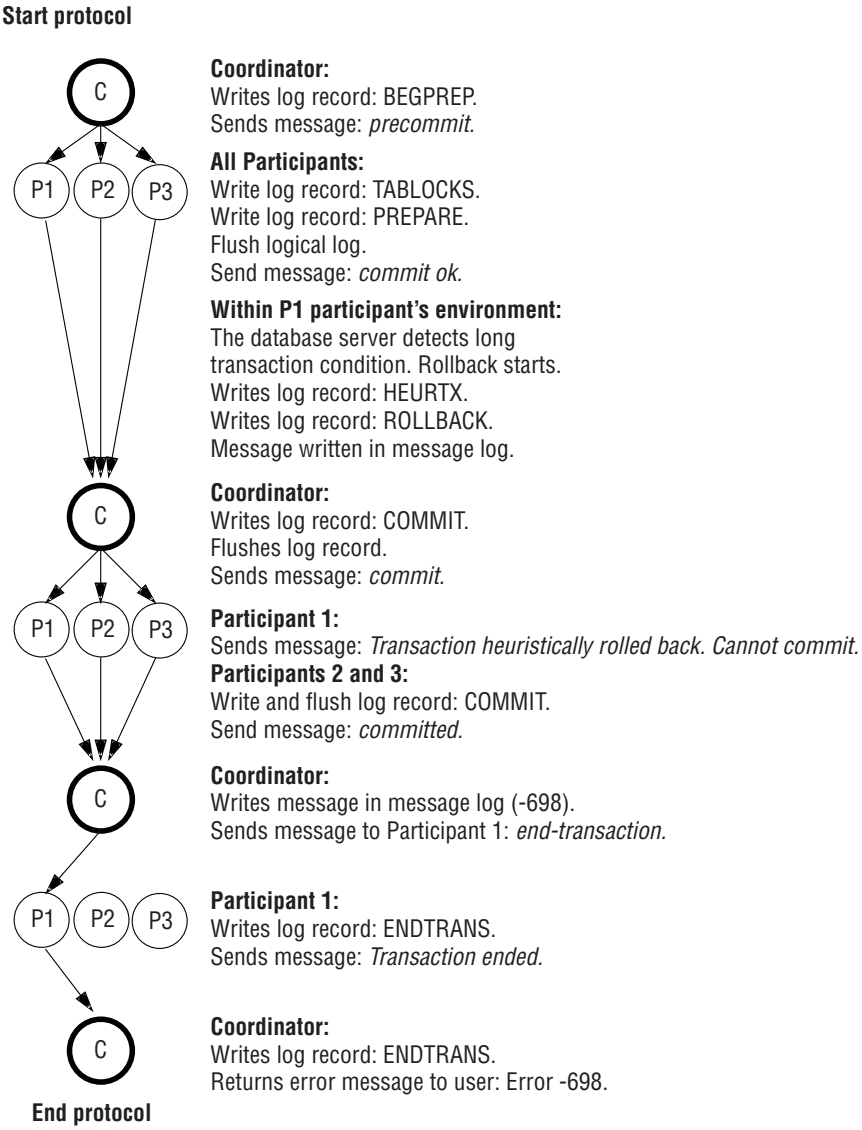


Figure 25-4. Logical-Log Records Written During a Heuristic Rollback

Logical-Log Records Written After a Heuristic End Transaction

Figure 25-5 on page 25-19 illustrates the writing sequence of the logical-log records during a heuristic end transaction. The event is always the result of a database server administrator ending a transaction (see information on the **onmode** utility in the *IBM Informix Administrator's Reference*) at a participant database server after the participant has sent a `can commit` message. In Figure 25-5 on page 25-19, the heuristic end transaction is assumed to have occurred at the Participant 1 (P1) database server. The result is an inconsistently implemented transaction. See "The Heuristic End-Transaction Scenario" on page 25-12.

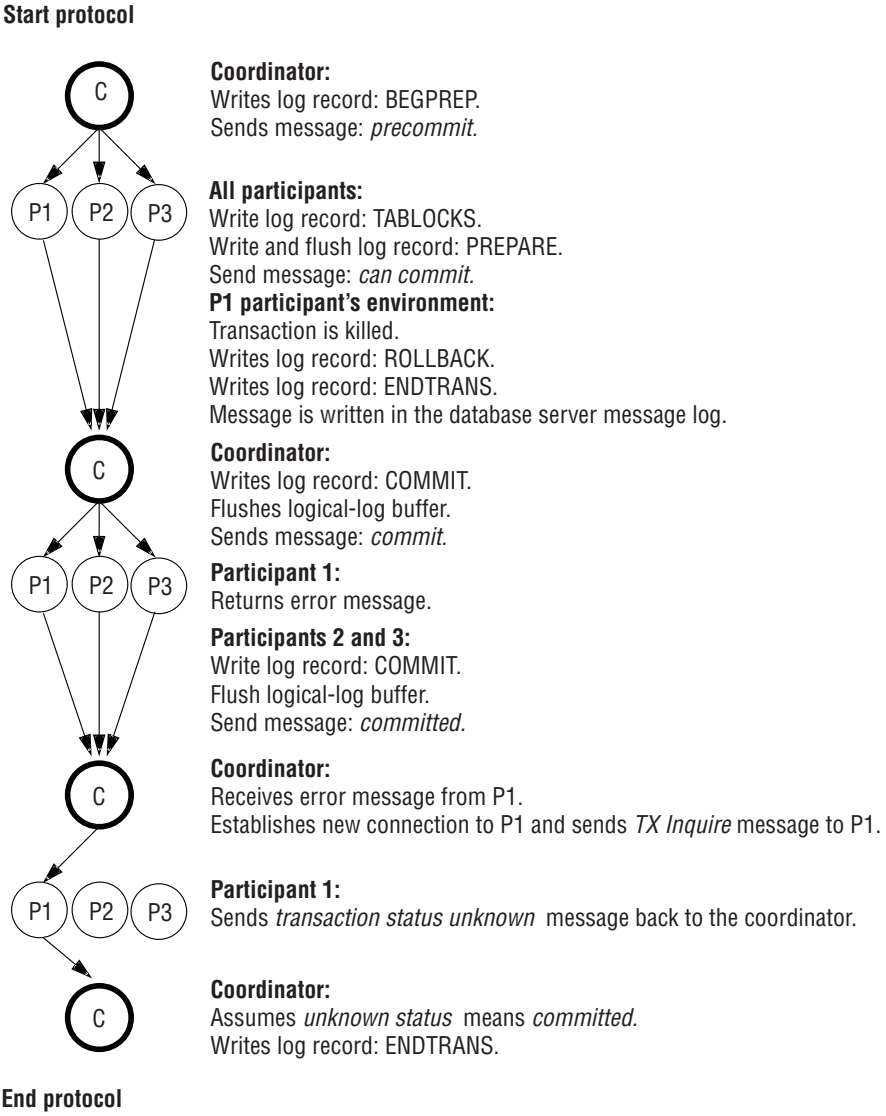


Figure 25-5. Logical-Log Records Written During a Heuristic End Transaction

Configuration Parameters Used in Two-Phase Commits

The following two configuration-file parameters are specific to distributed environments:

- DEADLOCK_TIMEOUT
- TXTIMEOUT

Although both parameters specify time-out periods, the two are independent. For more information on these configuration parameters, see the *IBM Informix Administrator's Reference*.

Function of the DEADLOCK_TIMEOUT Parameter

If a distributed transaction is forced to wait longer than the number of seconds specified by DEADLOCK_TIMEOUT for a shared-memory resource, the thread that owns the transaction assumes that a multiserver deadlock exists. The following error message is returned:

```
-154 ISAM error: deadlock timeout expired - Possible deadlock.
```

The default value of DEADLOCK_TIMEOUT is 60 seconds. Adjust this value carefully. If you set it too low, individual database servers end transactions that are not deadlocks. If you set it too high, multiserver deadlocks could reduce concurrency.

Function of the TXTIMEOUT Parameter

The TXTIMEOUT configuration parameter is specific to the two-phase commit protocol. It is used only if communication between a transaction coordinator and participant has been interrupted and needs to be reestablished.

The TXTIMEOUT parameter specifies a period of time that a participant database server waits to receive a *commit* instruction from a coordinator database server during a distributed transaction. If the period of time specified by TXTIMEOUT elapses, the participant database server checks the status of the transaction to determine if the participant should initiate automatic participant recovery.

TXTIMEOUT is specified in seconds. The default value is `300` (five minutes). The optimal value for this parameter varies, depending on your specific environment and application. Before you modify this parameter, read the discussion "How the Two-Phase Commit Protocol Handles Failures" on page 25-6.

Heterogeneous Commit Protocol

Used in the context of Informix database servers, the term heterogeneous environment refers to a group of database servers in which at least one of the database servers is not an Informix database server. Heterogeneous commit ensures the all-or-nothing basis of distributed transactions in a heterogeneous environment.

Unlike the two-phase commit protocol, the heterogeneous commit protocol supports the participation of a non-Informix participant. The non-Informix participant, called a gateway participant, must communicate with the coordinator through an IBM Informix gateway.

The database server uses heterogeneous commit protocol when the following criteria are met:

- Heterogeneous commit is enabled. (That is, the HETERO_COMMIT configuration parameter is set to `1`.)
- The coordinator of the commit is a Version 7.2 or later IBM Informix Dynamic Server.
- The non-Informix participant communicates with the Informix database server through an Informix gateway.
- At most, one non-Informix participant performs an update within a single transaction.

Figure 25-6 illustrates this scenario.

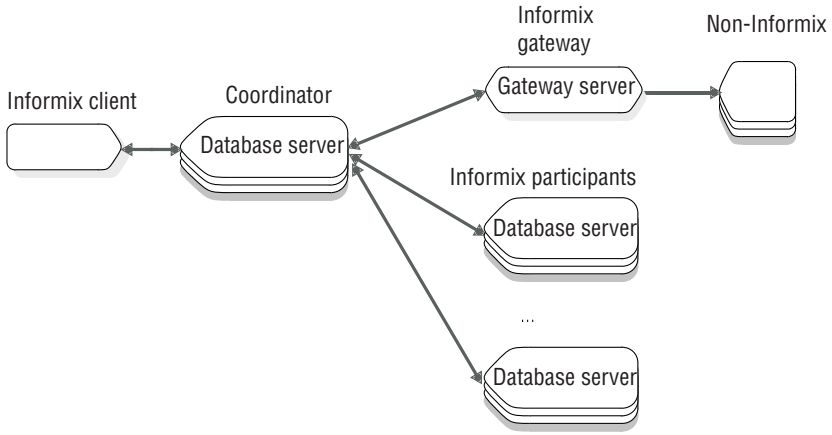


Figure 25-6. Configuration That Requires Heterogeneous Commit for Distributed Transactions

Gateways That Can Participate in a Heterogeneous Commit Transaction

An IBM Informix gateway acts as a bridge between an Informix application (in this case, a database server) and a non-IBM Informix database server. An IBM Informix gateway allows you to use an IBM Informix application to access and modify data that is stored in non-Informix databases.

The following table lists the gateways and corresponding database servers that can participate in a transaction in which the database server uses the heterogeneous commit protocol.

Table 25-1. Gateways and corresponding database servers/heterogeneous commit transaction

Gateway	Database Servers
IBM Informix Enterprise Gateway with DRDA	IBM DB2®, OS/400®, SQL/DS™
IBM Informix Enterprise Gateway for EDA/SQL	EDA/SQL
IBM Informix Enterprise Gateway Manager	Any database server with ODBC connectivity

Enabling and Disabling of Heterogeneous Commit

Use a text editor or ISA to change the HETERO_COMMIT configuration parameter which enables or disables heterogeneous commit: The change takes effect when you shut down and restart the database server.

When you set HETERO_COMMIT to `1`, the transaction coordinator checks for distributed transactions that require the use of heterogeneous commit. When the coordinator detects such a transaction, it automatically executes the heterogeneous commit protocol.

If you set HETERO_COMMIT to `0` or any number other than 1, the transaction coordinator disables the heterogeneous commit protocol. The following table summarizes which protocol the transaction coordinator uses, heterogeneous commit or two-phase commit, to ensure the integrity of a distributed transaction.

HETERO_COMMIT Setting	Gateway Participant Updated	Database Server Protocol
Disabled	No	Two-phase commit
Disabled	Yes	Two-phase commit
Enabled	No	Two-phase commit
Enabled	Yes	Heterogeneous commit

How Heterogeneous Commit Works

The heterogeneous commit protocol is a modified version of the standard two-phase commit protocol. The postdecision phase in the heterogeneous commit protocol is identical to the postdecision phases in the two-phase commit protocol. The precommit phase contains a minor modification, and a new phase, called the gateway commit phase, is added to the heterogeneous commit protocol.

The following sections explain the modification to the precommit phase and the gateway commit phase. For a detailed explanation of the postdecision phases, see "Postdecision Phase" on page 25-6.

Precommit Phase

The coordinator directs each update participant (except the gateway participant) to prepare to commit the transaction.

If the updates satisfy all deferred constraints, all participants (except the gateway participant) return messages to the coordinator indicating that they can commit their piece of work.

Gateway Commit Phase

If all participants successfully return a message indicating that they are prepared to commit, the coordinator sends a commit message to the gateway. The gateway in turn sends a response to the coordinator indicating whether the gateway committed its piece of the transaction. If the gateway commits the transaction, the coordinator decides to commit the entire transaction. Figure 25-7 on page 25-23 illustrates this process.

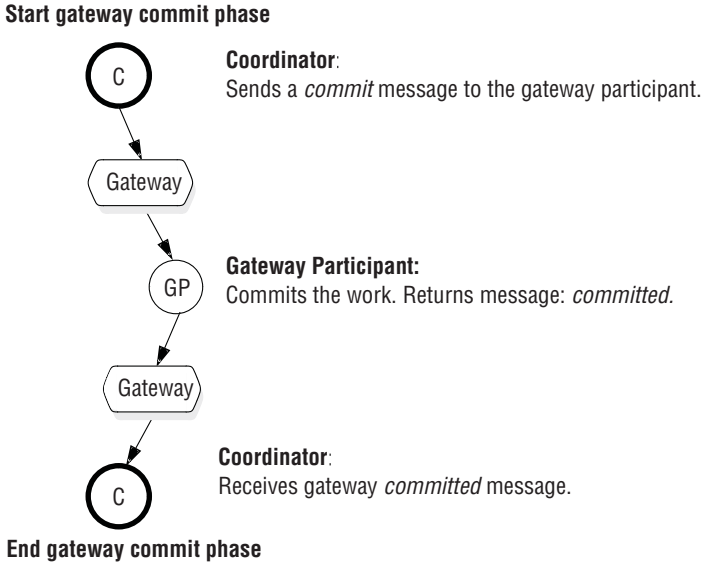


Figure 25-7. Heterogeneous Commit Phase That Results in a Committed Transaction

If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction, as Figure 25-7 illustrates.

Heterogeneous Commit Optimization

The database server optimizes the heterogeneous commit protocol when the only participant that receives an update is a non-Informix database. In this case, the coordinator sends a single commit message to all participants without invoking the heterogeneous commit protocol.

Implications of a Failed Heterogeneous Commit

At any time during a distributed transaction that the database server processes using heterogeneous commit, the coordinator or any number of participants can fail. The database server handles these failures in the same way as in the two-phase commit protocol, except in certain instances. The following sections examine these special instances in detail.

Database Server Coordinator Failure

The consistency of data after a coordinator failure depends on the point in the heterogeneous commit process at which the coordinator fails. If the coordinator fails before sending the commit message to the gateway, the entire transaction is stopped upon recovery, as is the case with two-phase commit.

If the coordinator fails after it writes the commit log record, the entire transaction is committed successfully upon recovery, as is the case with two-phase commit.

If the coordinator fails after it sends the commit message to the gateway but before it writes the commit log record, the remote Informix database server sites in the transaction are stopped upon recovery. This can result in inconsistencies if the gateway received the commit message and committed the transaction.

The following table summarizes these scenarios.

Point of Database Server Coordinator Failure	Expected Result
After the coordinator writes the PREPARE log record and before the gateway commit phase	Data consistency is maintained.
After the coordinator sends a commit message to the gateway but before it receives a reply	Data is probably inconsistent. No indication of probable data inconsistency from the coordinator.
After gateway commit phase but before the coordinator writes a COMMIT record to the logical log	Data consistency is lost. No indication of data inconsistency from the coordinator.

Participant Failure

Whenever a participant in a distributed transaction that uses the heterogeneous protocol fails, the coordinator sends the following error message:

```
-441    Possible inconsistent data at the target DBMS name due to an
aborted commit.
```

In addition, the database server sends the following message to the message log:

```
Data source accessed using gateway name might be in an inconsistent state.
```

A participant failure is not limited to the failure of a database server or gateway. In addition, a failure of the communication link between the coordinator and the gateway is considered a gateway failure. The gateway terminates if a link failure occurs. The gateway must terminate because it does not maintain a transaction log and therefore cannot reestablish a connection with the coordinator and resume the transaction. Because of this restriction, some scenarios exist in which a gateway failure might leave data in an inconsistent state. The following table summarizes these scenarios.

Point of Participant Failure	Expected Result
After participant receives `commit transaction` message from coordinator, but before participant performs commit	Data consistency is maintained.
After participant receives `commit transaction` message from coordinator and commits the transaction, but before the participant replies to coordinator	Data is inconsistent.
After participant commits the transaction and sends a reply to coordinator	If the communications link fails before the coordinator receives the reply, then data is inconsistent. If the coordinator receives the reply, then data is consistent (provided the coordinator does not fail before writing the COMMIT record).

The recovery procedure that the database server follows when a participant fails is identical to the procedure that is followed in two-phase commit. For more information on this procedure, see "Participant Failure."

Interpretation of Heterogeneous Commit Error Messages

When the database server fails to process a distributed transaction using heterogeneous commit, it returns one of the two error messages that are discussed in the following sections.

Application Attempts to Update Multiple Gateway Participants:

If your client application attempts to update data at more than one gateway participant when HETERO_COMMIT is set to 1, the coordinator returns the following error message:

```
-440 Cannot update more than one non-Informix DBMS within a transaction.
```

If you receive this error message, rewrite the offending application so that it updates at most one gateway participant in a single distributed transaction.

Failed Attempt to Commit Distributed Transaction Using Heterogeneous Commit:

The database server can fail to commit a distributed transaction while it is using the heterogeneous protocol for one or more of the following reasons:

- Communication error
- Site failure
- Gateway failure
- Other unknown error

When such a failure occurs, the coordinator returns the following message:

```
-441    Possible inconsistent data at the target DBMS name due to
an aborted commit.
```

After the database server sends this message, it rolls back all update sites that are participating in the transaction, with the possible exception of the work done at the site of the gateway participant. The gateway participant might have committed its updates if the failure occurred after the gateway participant processed the commit message. If the gateway participant committed the updates, you must manually rollback these updates.

Chapter 26. Recovering Manually from Failed Two-Phase Commit

In This Chapter

Distributed transactions follow the two-phase commit protocol. Certain actions occur independently of the two-phase commit protocol and cause the transaction to be inconsistently implemented. (See "Independent Actions" on page 25-8.) In these situations, it might be necessary to recover manually from the transaction.

This chapter describes the following topics:

- Determining if you need to recover manually from an inconsistently implemented two-phase commit transaction
- Performing a manual recovery

Determining If Manual Recovery Is Required

The following list outlines the steps in the procedure to determine database consistency and to correct the situation if required.

To determine database consistency

1. Determine whether a transaction was implemented inconsistently.
2. Determine if the networked database system contains inconsistent data.
3. Decide if action to correct the situation is required.

Each of these steps is described in the following sections.

Determining If a Transaction Was Implemented Inconsistently

Your first task is to determine whether the transaction was implemented inconsistently as a result of an independent action.

Global Transaction Ended Prematurely

If you executed an **onmode -z** command to end the global transaction on the coordinator, the transaction might be inconsistently implemented. (For an explanation of how this situation can arise, see "Independent Actions That Result in an Error Condition" on page 25-9.) You can check for an inconsistent transaction by first examining the database server message log for the coordinator. Look for the following error message:

```
-716 Possible inconsistent transaction.
Unknown servers are server-name-list.
```

This message lists all the database servers that were participants. Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic End Transaction

If you executed an **onmode -Z** *address* command to end a piece of work performed by a participant, *and* the coordinator decided to commit the transaction, the transaction is implemented inconsistently. (For a description of this scenario,

see "The Heuristic End-Transaction Scenario" on page 25-12.) Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic Rollback

You can determine the specific database server participants affected by a heuristic decision to roll back a transaction in the following ways:

- Examine the return code from the COMMIT WORK statement in the application.

  The following message indicates that one of the participants performed a heuristic rollback:

  ```
  -698 Inconsistent transaction. Number and names of servers rolled back.
  ```

- Examine the messages in the database server message-log file.

  If a database inconsistency is possible because of a heuristic decision at a participating database server, the following message appears in the database server message-log file of the coordinator:

  ```
  Mixed transaction result. (pid=nn user=user_id)
  ```

  This message is written whenever error -698 is returned. Associated with this message is a list of the participant database servers where the transaction was rolled back. This is the complete list. The list that appears with the -698 error message could be truncated if a large number of participants rolled back the transaction.

- Examine the logical log for each participant.

  If at least one participant rolls back its piece of work and one participant commits its piece of work, the transaction is implemented incorrectly.

Determining If the Distributed Database Contains Inconsistent Data

If you determine that a transaction was inconsistently implemented, you must determine what this situation means for your distributed database system. Specifically, you must determine if data integrity has been affected.

A transaction that is inconsistently implemented causes problems whenever the piece of work rolled back by one participant is dependent on a piece of work that was updated by another participant. It is impossible to define these dependencies with SQL because distributed transactions do not support constraints that reference data at multiple database servers. The pieces of work are independent (no dependencies exist) only if the data could have been updated in two independent transactions. Otherwise, the pieces of work are considered to be dependent.

Before you proceed, consider the transaction that caused the error. Are the pieces of data that were updated and rolled back dependent on one another? Multiple updates might be included in a single transaction for reasons other than maintaining data integrity. For example, three possible reasons are as follows:

- Reduced transaction overhead
- Simplified coding
- Programmer preference

Verify also that every participant database server that is assumed to have committed the transaction actually modified data. A read-only database server might be listed as a participant that committed a transaction.

If an inconsistent transaction does not lead to a violation of data integrity, you can quit the procedure at this point.

Obtaining Information from the Logical Log

To determine if data integrity has been affected by an inconsistently implemented global transaction, you need to reconstruct the global transaction and determine which parts of the transaction were committed and which were rolled back. Use the **onlog** utility to obtain the necessary information. The procedure is as follows:

1. Reconstruct the transaction at the participant that contains the HEURTX record.
   a. A participant database server logical log is the starting point for your information search. Each record in the log has a local transaction identification number (**xid**). Obtain the **xid** of the HEURTX record.
   b. Use the local **xid** to locate all associated log records that rolled back as part of this piece of work.
2. Determine which database server acted as coordinator for the global transaction.
   a. Look for the PREPARE record on the participant that contains the same local **xid**. The PREPARE record marks the start of the two-phase commit protocol for the participant.
   b. Use the **onlog -l** option to obtain the long output of the PREPARE record. This record contains the global transaction identifier (GTRID) and the name of the coordinating database server. For information about GTRID, see "Obtaining the Global Transaction Identifier."
3. Obtain a list of the other participants from the coordinator log.
   a. Examine the log records on the coordinator database server. Find the BEGPREP record.
   b. Examine the long output for the BEGPREP record. If the first 32 bytes of the GTRID in this record match the GTRID of the participant, the BEGPREP record is part of the same global transaction. Note the participants displayed in the ASCII part of the BEGPREP long output.
4. Reconstruct the transaction at each participant.
   a. At each participant database server, read the logical log to find the PREPARE record that contains the GTRID associated with this transaction and obtain the local **xid** for the piece of work performed by this participant.
   b. At each participant database server, use the local **xid** to locate all logical-log records associated with this transaction (committed or rolled back).

After you follow this procedure, you know what all the participants for the transaction were, which pieces of work were assigned to each participant, and whether each piece of work was rolled back or committed. From this information, you can determine if the independent action affected data integrity.

Obtaining the Global Transaction Identifier

When a global transaction starts, it receives a unique identification number called a global transaction identifier (GTRID). The GTRID includes the name of the coordinator. The GTRID is written to the BEGPREP logical-log record of the coordinator and the PREPARE logical-log record of each participant.

To see the GTRID, use the **onlog -l** option. The GTRID is offset 20 bytes into the data portion of the record and is 144 bytes long. Figure 26-1 on page 26-4 shows

the **onlog -l** output for a BEGPREP record. The coordinator is **chrisw**.

```
4a064    188  BEGPREP  4        0  4a038       0    1
         000000bc 00000043 00000004 0004a038 .......C .......8
         00087ef0 00000002 63687269 73770000 ..~..... chrisw..
         00000000 00000000 00000000 00087eeb ........ ......~.
         00006b16 00000000 00000000 00000000 ..k..... ........
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000001 6a756469 74685f73 ........ judith_s
         6f630000 736f6374 63700000          oc..soct cp..
```

Figure 26-1. Output of the onlog -l Option for a BEGPREP Record

The first 32 bytes of the GTRID are identical for the BEGPREP record on the coordinator and the PREPARE records on participants, which are part of the same global transaction. For example, compare the GTRID for the PREPARE record in Figure 26-2 with that of the BEGPREP record in Figure 26-1.

```
c7064    184  PREPARE  4        0  c7038    chrisw
         000000b8 00000044 00000004 000c7038 .......D ......p8
         00005cd6 00000002 63687269 73770000 ......  chrisw..
         00000000 00000000 00000069 00087eeb ........ ...i..~.
         00006b16 00000000 00000010 00ba5a10 ..k..... ......Z.
         00000002 00ba3a0c 00000006 00000000 ......:. ........
         00ba5a10 00ba5a1c 00000000 00000000 ..Z...Z. ........
         00ba3a0e 00254554 00ba2090 00000001 ..:..%ET .. .....
         00000000 00ab8148 0005fd70 00ab8148 .......H ...p...H
         0005fe34 0000003c 00000000 00000000 ...4...< ........
         00000000 00ab80cc 00000000 00ab80c4 ........ ........
         00ba002f 63687269 73770000 00120018 .../chrisw......
         00120018 00ba0000                   ........
```

Figure 26-2. Output of the onlog -l Option for a PREPARE Record

Deciding If Action Is Needed to Correct the Situation

If an inconsistent transaction creates an inconsistent database, the following three options are available to you:

- Leave the networked database in its inconsistent state.
- Remove the effects of the transaction wherever it was committed, thereby rolling back the entire transaction.
- Reapply the effects of the transaction wherever it was rolled back, thereby committing the transaction.

You can leave the database in its inconsistent state if the transaction does not significantly affect database data. You might encounter this situation if the application that is performing the transaction can continue as it is, and you decide that the price (in time and effort) of returning the database to a consistent state by either removing the effects or reapplying the transaction is too high.

You do not have to reach this decision immediately. You can use the methods described in the following paragraphs to determine what data the transaction was updating and which records are affected.

As you make your decision, consider that no automatic process or utility can perform a rollback of a committed transaction or can commit part of a transaction that has been rolled back. The following paragraphs describe how to look through the database server message log and the logical log to locate affected records. Without detailed knowledge of the application, messages are not enough to determine what has happened. Based on your knowledge of your application and your system, you must determine whether to roll back or to commit the transaction. You must also program the compensating transaction that will perform the rollback or the commit.

Example of Manual Recovery

This example illustrates the kind of work that is involved in manual recovery. The following SQL statements were executed by user **nhowe**. Error -698 was returned.

```
dbaccess
CREATE DATABASE tmp WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
CREATE DATABASE tmp@apex WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
DATABASE tmp;
BEGIN WORK;
INSERT INTO t VALUES (2);
INSERT INTO tmp@apex:t VALUES (2);
COMMIT WORK;
### return code -698
```

The following excerpt is taken from the logical log at the current database server:

```
addr      len   type      xid      id   link
.....
17018    16         CKPOINT  0       0     13018    0

18018    20         BEGIN    2       1     0        08/27/91 10:56:57
   3482     nhowe

1802c    32         HINSERT  2       0     18018    1000018  102
   4

1804c    40         CKPOINT  0       0     17018    1

   begin    xid    id addr     user

   1        2          1  1802c    nhowe

19018    72         BEGPREP  2       0     1802c   6d69    1

19060    16         COMMIT   2       0     19018   08/27/91 11:01:38

1a018    16         ENDTRANS 2       0     19060   580543
```

The following excerpt is taken from the logical log at the database server **apex:**

```
addr      len   type      xid      id   link
.....
16018    20         BEGIN    2            1   0        08/27/91
   10:57:07 3483     pault

1602c    32         HINSERT  2       0     16018    1000018  102
   4

1604c    68         PREPARE  2       0     1602c      eh
```

```
17018    16         HEURTX   2       0     1604c          1

17028    12         CLR      2       0     1602c

17034    16         ROLLBACK 2       0     17018    08/27/91 11:01:22

17044    40         CKPOINT  0       0     15018    1

      begin    xid      id addr     user
      1        2        1  17034    --------

18018    16         ENDTRANS 2       0     17034     8806c3
   ....
```

First, you would try to match the transactions in the current database server log with the transactions in the **apex** database server log. The BEGPREP and PREPARE log records each contain the GTRID. You can extract the GTRID by using **onlog -l** and looking at the data portion of the BEGPREP and PREPARE log records. The GTRID is offset 22 bytes into the data portion and is 68 bytes long. A more simple, though less precise, approach is to look at the time of the COMMIT or ROLLBACK records. The times should be close, although there is a slight delay because of the time taken to transmit the commit (or rollback) message from the coordinator to the participant. (This second approach lacks precision because concurrent transactions could commit at the same time although concurrent transactions from one coordinator would probably not commit at the same time.)

To correct this sample situation

1. Find all records that were updated.
2. Identify their type (insert, delete, update) using **onlog** and the table of record types.
3. Use the **onlog -l** output for each record to obtain the local **xid**, the tblspace number, and the rowid.
4. Map the tblspace number to a table name by comparing the tblspace number to the value in the **partnum** column of the **systables** system catalog table.
5. Using your knowledge of the application, determine what action is required to correct the situation.

In this example, the time stamps on the COMMIT and ROLLBACK records in the different logs are close. No other active transactions introduce the possibility of another concurrent commit or rollback. In this case, an insert (HINSERT) of assigned rowid 102 hex (258 decimal) was committed on the current database server. Therefore, the compensating transaction is as follows:

```
DELETE FROM t WHERE rowid = 258
```

Part 6. Automatic Monitoring and Corrective Actions

This chapter provides information on the SQL administration API, the Scheduler, and functionality for performing Query Drill-Down query drill-down to analyze SQL statement history.

See Chapter 27, “Overview of Automatic Monitoring and Corrective Actions,” on page 27-1, Chapter 31, “Remote Administration with SQL Administration API Commands,” on page 31-1, Chapter 30, “The Scheduler,” on page 30-1, and Chapter 32, “Query Drill-Down,” on page 32-1.

Chapter 27. Overview of Automatic Monitoring and Corrective Actions

You can use the SQL administration API, the Scheduler, information stored in the **sysadmin** database, and Query Drill-Down functionality to manage automatic maintenance, monitoring, and administrative tasks. These components of Dynamic Server enable you to simplify the collection of information and maintenance of the server in complex systems.

SQL administration API
The SQL administration API enables you to perform remote administration using specific SQL commands. For more information, see Chapter 31, "Remote Administration with SQL Administration API Commands," on page 31-1.

Scheduler
The Scheduler enables the database server to execute database functions and procedures at predefined times or as determined internally by the server. The functions and procedures collect information and monitor and adjust the server, using an SQL-based administrative system and a set of tasks. For more information, see Chapter 30, "The Scheduler," on page 30-1.

sysadmin database
The **sysadmin** database contains tables that store task properties. You use the task properties (not configuration parameters) to define the information that the Scheduler collects and the statements that the Scheduler runs. For more information on this database, see Chapter 28, "The sysadmin Database," on page 28-1.

The **command_history** table in the **sysadmin** database contains historical information about the SQL administration API commands executed on this data server. For more information, see Chapter 29, "The command_history Table," on page 29-1.

Query Drill-Down functionality for analyzing SQL statement history
Query Drill-Down functionality provides statistical information on recently executed SQL statements, enabling you to track the performance on individual SQL statements and analyze statement history. For more information, see Chapter 32, "Query Drill-Down," on page 32-1.

You can use the SQL administration API and the Scheduler on the primary server of an HDR pair of servers.

Because SQL administration API operations occur entirely in SQL, client tools can use these features to administer the database server.

You can also use a PHP-based Web browser administration tool, the OpenAdmin Tool for IDS, to administer multiple database server instances from a single location. Some tasks you can perform with OpenAdmin include:

- Defining and managing automated tasks through the SQL administration API
- Creating and displaying performance histograms for analysis and tuning
- Monitoring high availability solutions that include HDR, shared disk secondary servers, and remote standalone secondary servers.

You can easily plug in your own extensions to OpenAdmin to create the functionality you need.

OpenAdmin is an open-source program that you can download from this Web site: http://www.ibm.com/software/data/informix/downloads.html

Increased Storage Space Requirements

If you use the SQL administration API and the Scheduler, and perform Query Drill-Down, you need disk space for these features and for additional the storage needed for historical tables.

If you turn Query Drill-Down functionality on, the default amount of memory required is two megabytes. You can tune this memory by using the SQL administration API while the system is online or by modifying the SQLTRACE configuration parameter in the ONCONFIG file.

Tasks that function as sensors by collecting and saving information in tables can consume disk space in the database server. You can use the following formula to estimate the disk usage for one sensor:

*Number of rows collected * size of the row collected * the frequency of data collection per day * the retention period*

Repeat this estimate for all sensors and you can determine a close estimate of the space required.

You can disable the sensors that collect and save historical information. However, disabling these sensors can effect other database server functions.

Chapter 28. The sysadmin Database

The **sysadmin** database, which contains tables that store task properties, is a logged database. You use the task properties (not configuration parameters) to define the information that the Scheduler collects and the statements the Scheduler executes.

The **sysadmin** database also contains:

- The built-in **task()** function
- The built-in **admin()** function
- The **command_history** table, which contains information about the commands that the SQL administration API ran

Important: Never drop or alter the **sysadmin** database, because several database server components use the database.

The **sysadmin** database is created in the root dbspace by default. If you want to move the database, see "Moving the sysadmin Database to a New Database Space" on page 29-2

sysadmin Database Tables

The **sysadmin** database contains the following tables:

Table 28-1. sysadmin Database Tables

Table	Description
PH_ALERT	Contains a list of errors, warnings, or information messages that must be monitored.
PH_GROUP	Contains a list of group names. Each task is a member of a group.
PH_RUN	Contains information about how and when each task was executed.
PH_TASK	Lists tasks and contains information about how and when the database server will execute each task.
PH_THRESHOLD	Contains a list of thresholds that you defined. If a threshold is met, the task can decide to take a different action, such as inserting an alert in the PH_ALERT table.

Each row in the **ph_task** table is a separate *task* (defined as a single monitoring event) and each column is a task property. The task properties indicate such items as when to run an SQL statement, stored procedure, or UDR and how to handle the task. An example of a task is the automatic running of a particular job every Monday at midnight.

For details about these tables, see the *IBM Informix Dynamic Server Administrator's Reference*.

SQL Administration API Functions

Two built-in SQL administration API functions, **task()** and **admin()**, are available in the **sysadmin** database.

By default, only user **informix**, can connect to the **sysadmin** database. If user **root** or a member of the DBSA group is granted privileges to connect to the **sysadmin** database, then user root or a member of the DBSA group can also invoke the SQL administration API **task()** and **admin()** functions.

The **task()** and **admin()** functions provide the same functionality; they differ only in their return code. The **task()** function returns a string that describes the results of the command. The **admin()** function returns an integer.

You use SQL to define or modify these functions. For example, you can define a task that creates a dbspace, as follows:

```
EXECUTE FUNCTION admin('create dbspace', 'dbspace2', '/work/dbspace2', "20 MB");
```

For information on defining **admin()** and **task()** functions and examples, see the *IBM Informix Dynamic Server Administrator's Reference*.

Chapter 29. The command_history Table

The **command_history** table contains a list of all commands that the SQL administration API ran. The table also shows the results of the commands. This table, which is in the **sysadmin** database, is a RAW (nonlogged) table.

The **command_history** table shows if an administrative task was executed through an **admin()** or **task()** function and displays information about the user who executed the command, the time the command was executed, the command, and the message returned when the database server completed running the command.

The following table shows sample commands and the associated results in a sample **command_history** table. For a description of all information in the **command_history** table, see the *IBM Informix Dynamic Server Administrator's Reference*.

Table 29-1. Example of some Information in a command_history Table

Command Executed	Sample Returned Messages
set sql tracing on	SQL tracing on with 1000 buffers of 2024 bytes.
create dbspace	Space 'space12' added.
checkpoint	Checkpoint completed.
add log	Added 3 logical logs to dbspace logdbs.

To display the command history, run this SQL statement:

```
SELECT * from command_history
```

Tasks in the **command_history** table are automatically removed after a fixed period of time. You can modify this time period by changing information in the COMMAND HISTORY RETENTION row in the **ph_threshold** table. The COMMAND HISTORY RETENTION parameter sets the length of time rows should remain in the **command_history** table.

You can use SQL commands like `delete` or `truncate table` to manually remove data from this table. You can also create a task in the **ph_task** table to purge data from the **command_history** table. The following example shows a task that monitors the amount of data in the **command_history** table and purges data when it becomes too old.

```
INSERT INTO ph_task
( tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency )
VALUES
("mon_command_history",
"TASK",
"TABLES",
"Monitor how much data is kept in the command history table",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY RETENTION' ) ",
DATETIME(02:00:00) HOUR TO SECOND,
NULL,
INTERVAL ( 1 ) DAY TO DAY);
```

Moving the sysadmin Database to a New Database Space

The **sysadmin** database is created in the root dbspace by default. If the root dbspace does not have enough space for storing task properties and command history information, you can move the **sysadmin** database to a different dbspace by using the `"reset sysadmin"` SQL administration API command. This command removes the **sysadmin** database from root dbspace and recreates it in the specified dbspace.

To move the **sysadmin** database:

1. Make sure the following message has appeared in the online message log after server startup:

   ```
   SCHAPI: Started 2 dbWorker threads.
   ```

2. If necessary, create a new dbspace for the **sysadmin** database, for example *new_dbspace*.
3. As user **informix**, run the following commands:

   ```
   dbaccess sysadmin -
   execute function task("reset sysadmin", "new_dbspace");
   ```

   where *new_dbspace* is the name of the dbspace that will store the **sysadmin** database.

   The command returns the following message:

   ```
   SCHAPI: 'sysadmin' database will be moved to 'new_dbspace'.
   See online message log.
   ```

   The internal thread, **bld_sysadmin**, waits up to five minutes to obtain exclusive access to the **sysadmin** database. The progress of the **bld_sysadmin** thread is logged in the online message log.
4. Terminate the dbaccess session with the **close database** statement.

If this operation completes successfully, the **sysadmin** database is dropped and recreated in the new dbspace. The Scheduler and dbWorker threads are started automatically.

Chapter 30. The Scheduler

The Scheduler manages and executes scheduled maintenance, monitoring, and administration tasks.

This tool enables you to monitor activities (for example, space management or automatically backing up any new log data at timed intervals since the last log backup) and create corrective actions that run automatically.

The Scheduler manages:

- Tasks, which provide the means for running a specific job at a specific time or interval.
- Sensors, which collect and save information
- Startup tasks, which run only once when the database server starts
- Startup sensors, which run only once when the database starts

A set of task properties, which define what needs to be collected or executed, control the Scheduler. The task properties are stored in the **ph_task** table in the **sysadmin** database. Each row in this table is a separate task and each column is a task property. The task properties indicate to the system such things as when to run an SQL statement, stored procedure, or function and how to handle the task.

For example, you could define tasks to check free log space every hour from 9:00:00 to 19:00:00 daily.

Only task properties, not configuration parameters, define what the Scheduler collects and executes. The Scheduler executes tasks at predefined times or as determined internally as required by the database server.

The Scheduler provided with Dynamic Server contains the tasks shown in the following table.

Table 30-1. Built-In Tasks

Task	Description
Auto Update Statistics Evaluation	This task analyzes all the tables in all logged databases, identifies the tables whose distributions need to be updated, and generates UPDATE STATISTICS statements for those tables, based on the current automatic update statistics (AUS) policies.
Auto Update Statistics Refresh	This task recalculates table statistics for use by the query optimizer. Statements from the Auto Update Statistics Evaluation task are executed in their order of priority during a recurring time interval that the DBA specifies for this task.
mon_command_history	Purges the command history table.
mon_config	Saves any changes in the ONCONFIG file.
mon_config_startup	Saves the ONCONFIG file on every server startup.
mon_profile	Saves server profile information.

Table 30-1. Built-In Tasks (continued)

Task	Description
mon_vps	Collects virtual processor information.
mon_checkpoint	Saves information about checkpoints.
mon_table_profile	Saves table profile information, including the total number of updates, inserts, and deletes that occurred on this table.
mon_table_names	Saves the table names along with their creation time.
mon_users	Saves profile information about each user.
check_backup	Checks to ensure that backups have run.

You can modify these tasks and set up more tasks. See "Setting Up Tasks" and "Modifying Tasks" on page 30-6.

Setting Up Tasks

Chapter 30, "The Scheduler," on page 30-1 manages and executes scheduled maintenance, monitoring, and administration tasks. You can set up new tasks by inserting rows into the **ph_task** table. Do this when you want to create new monitoring activities or corrective actions that run automatically.

The **ph_task** table lists tasks and contains information about how and when the database server will execute each task.

When the database server executes a task, the server invokes the SQL object contained in the **tk_execute** column of the **ph_task** table. The **tk_execute** column can contain a user-defined function, a single SQL statement, or a multiple-statement prepared object that was created using the PREPARE SQL to enable the assembly of one or more SQL statements at runtime.

Procedure: To set up a task

1. Plan the task, including:
   - A description of the task you want to monitor.
   - The table where you want to store data.
   - The SQL command, stored procedure, or function to capture data.
   - Information on when and how often you want the task to run.
2. Use SQL to insert a new row into the **ph_task** table, as shown in the following examples. Include values that will be displayed in table columns.

The following example shows the code for a task that runs once a day at 2:00 A.M. to ensure that the **command_history** table contains only recent data. In this example, the definition of recent data is stored in a **Command History Interval** column in the **ph_threshold** table.

```
INSERT INTO ph_task
(
tk_name,
tk_group,
tk_description,
tk_type,
tk_execute,
tk_start_time,
tk_frequency
)
VALUES
(
"mon_command_history",
"TABLES",
"Monitor how much data is kept in the command history table",
"MONITOR",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY INTERVAL' ) ",
"2:00:00",
"1 0:00:00"
);
```

Figure 30-1. Sample Code for a Task that Runs Daily

The following example shows the code for the creation of a table called **mon_prof** that stores data and a view called **mon_profile**. In this example, data is collected every five minutes and the default scheduling policy (for example, 8:00 A.M. to 5:00 P.M. five days a week) is used. $DATA_SEQ_ID is a predefined value; a data sequence ID will replace this value.

```
INSERT INTO ph_task
(
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_frequency
)
VALUES
(
"Collect the general profile information",
"mon_prof",
"create table mon_prof (ID integer, number integer, value int8 );
   create view mon_profile as select ID, A.name,
   B.value from sysmaster:sysshmhdr A,
   mon_prof B where B.number = A.number ",
"insert into mon_prof select $DATA_SEQ_ID, number,
   value from ysmaster:sysshmhdr",
"0 0:05:00"
);
```

Figure 30-2. Sample Code for the Creation of a Table that Stores Data

The following example shows the code for the creation of a table, called **mon_chunkio**, that uses a stored procedure to collect and store data. This task instructs the database server to collect data every five minutes by executing a stored procedure called **chunkio**. The default scheduling policy (for example, 8:00 A.M. to 5:00 P.M. five days a week) is used, but the data expires and is deleted

when it becomes seven days old.

```
INSERT INTO ph_task
(
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_frequency,
tk_delete
)
VALUES
(
"Chunk I/O counts and space usage",
"mon_chunkio",
"create table mon_chunkio (ID integer, chunknum smallint,
   free integer, size integer, reads integer, pagereads integer,
   writes integer, pageswritten integer)",
"chunkio",
"0 0:05:00",
"7 0:00:00"
);
```

Figure 30-3. Sample Code for the Creation of a Table that Uses a Stored Procedure to Collect and Store Data

The following example shows the code for a sensor that tracks the startup environment of the database server.

```
INSERT INTO ph_task
(
tk_name,
tk_type,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete
)
VALUES
(
"mon_sysenv",
"STARTUP SENSOR",
"SERVER",
"Tracks the database servers startup environment.",
"mon_sysenv",
"create table mon_sysenv (ID integer, name varchar(250), value lvarchar(1024))",
"insert into mon_sysenv select $DATA_SEQ_ID, env_name, env_value FROM sysmaster:s
ysenv",
NULL,
NULL,
"0 0:01:00",
"60 0:00:00"
);

tk_id               7
tk_name             mon_sysenv
tk_description      Tracks the database servers startup environment.
tk_type             STARTUP SENSOR
tk_executing_sid    0
tk_sequence         0
tk_result_table     mon_sysenv
tk_create           create table mon_sysenv (ID integer, name varchar(250),
                    value lvarchar(1024))
tk_execute          insert into mon_sysenv select $DATA_SEQ_ID, env_name,
                    env_value FROM sysmaster:sysenv
tk_delete            60 00:00:00
tk_start_time
tk_stop_time
tk_frequency          0 00:01:00
tk_next_execution   2006-07-24 17:09:50
tk_total_executio+  0
tk_total_time       0.00
tk_monday           t
tk_tuesday          t
tk_wednesday        t
tk_thursday         t
tk_friday           t
tk_saturday         t
tk_sunday           t
tk_attributes       0
tk_group            SERVER
tk_enable           t
tk_priority         0
```

Figure 30-4. Sensor that Tracks the Startup Environment

The following example shows the code for a sensor that collects information about the amount of memory that is being used and stores the information in the

mon_memory_system table. If that table does not exist, the task creates it. This task, which runs every 30 minutes, deletes any data in the **mon_memory_system** table that has existed for more than 30 days.

```
INSERT INTO ph_task
(tk_name, tk_type, tk_group, tk_description, tk_result_table, tk_create,
tk_execute, tk_stop_time, tk_start_time, tk_frequency, tk_delete )
VALUES
("mon_memory_system",
"SENSOR",
"MEMORY",
"Server memory consumption",
"mon_memory_system",
"create table mon_memory_system (ID integer, class smallint, size int8,
     used int8, free int8 )",
"insert into mon_memory_system select $DATA_SEQ_ID, seg_class, seg_size,
     seg_blkused, seg_blkfree FROM sysmaster:sysseglst",
NULL,
NULL,
INTERVAL ( 30 ) MINUTE TO MINUTE,
INTERVAL ( 30 ) DAY TO DAY);
```

Figure 30-5. Sensor that collects information about the amount of memory being used.

Modifying Tasks

Use the rows in the tables that begin with the characters **ph_** to modify tasks.

To modify tasks:

1. Go to the **ph_task** or other table that you want to modify in the **sysadmin** database.
2. Manually change task information.

Chapter 31. Remote Administration with SQL Administration API Commands

You can use the SQL administration API to perform remote administration using various, specific SQL commands for tasks such as managing spaces, managing configuration, running routine jobs, and system validation.

You can use EXECUTE FUNCTION statements to invoke the built-in **admin( )** or **task( )** functions to accomplish administrative tasks that are equivalent to executing various administrative utilities of Dynamic Server.

In the EXECUTE FUNCTION statements, items in the argument list specify the utility and its command-line arguments. For example, the following SQL statement, which is equivalent to the **oncheck -ce** command, instructs the database server to check the extents:

```
EXECUTE FUNCTION admin('check extents');
```

Suppose that you want to increase the virtual memory the database server could use in an application. The application could execute this SQL statement:

```
EXECUTE FUNCTION task('add memory', '10 MB')
```

Suppose you want to drop all logical logs in the root chunk, except you do not want to drop the current logical log. You could use an Administrator API command within a SELECT statement as follows:

```
select task("drop log", number) from sysmaster:syslogfil
  where chunk = 1 and sysmaster:bitval(flags,"0x02")==0;
```

You must execute the **task()** and **admin()** functions against the **sysadmin** database.

For more information on all the SQL administration API commands you can use and information on defining **admin()** and **task()** functions, see the *IBM Informix Dynamic Server Administrator's Reference*.

Using onmode Syntax in SQL Administration API Commands

Use the SQL administration API to perform remote administration using various, specific SQL commands for tasks such as managing spaces, managing configuration, running routine jobs, and system validation. You can use EXECUTE FUNCTION statements to invoke the built-in **admin( )** or **task( )** functions in the **sysadmin** database to accomplish administrative tasks that are equivalent to executing commands using the **onmode** utility.

Only the DBSA can run the **task()** and **admin()** functions. However, only user **informix**, by default, can connect to the **sysadmin** database.

To use onmode syntax in your SQL administration API commands:

1. Create an EXECUTE FUNCTION statement to invoke a task.
2. In the arguments portion of the statement, include `onmode` and the applicable option without a hyphen in front of the option letter.

For example, if the **sysadmin** database is the current database, the following statement executes a task for gracefully shutting down the database server:

```
EXECUTE FUNCTION task ('onmode','k');
```

If the **sysadmin** database is not the current database, but you are a user who has permission to connect to the **sysadmin** database, you can execute the following command:

```
execute function sysadmin:task ('omode','k');
```

For information on **onmode** commands, see the *IBM Informix Dynamic Server Administrator's Reference*.

Chapter 32. Query Drill-Down

You can perform query drill-down to gather statistical information about each SQL statement executed on the system and analyze statement history.

The Query Drill-Down feature helps you answer questions such as:

- How long do SQL statements take?
- How many resources are individual statements using?
- How long did statement execution take?
- How much time was involved waiting for each resource?

The statistical information is stored in a circular buffer, which is an in-memory pseudo table, called **syssqltrace**, that is stored in the **sysmaster** database. You can dynamically resize the circular buffer.

By default this feature is turned off, but you can turn it on for all users or for a specific set of users. When this feature is enabled with its default configuration, the database server tracks the last 1000 SQL statements that ran, along with the profile statistics for those statements.

The memory required by this feature is large if you plan to keep a lot of historical information. The default amount of space required for SQL history tracing is two megabytes. You can expand or reduce the amount of storage according to your requirements. You can also disable SQL history tracing if you do not want to use memory for this.

Information displayed includes:

- The user ID of the user who ran the command
- The database session ID
- The name of the database
- The type of SQL statement
- The duration of the SQL statement execution
- The time this statement completed
- The text of the SQL statement or a function call list (also called *stack trace*) with the statement type, for example:

  ```
  procedure1() calls procedure2() calls procedure3()
  ```
- Statistics including the:
  - Number of buffer reads and writes
  - Number of page reads and writes
  - Number of sorts and disk sorts
  - Number of lock requests and waits
  - Number of logical log records
  - Number of index buffer reads
  - Estimated number of rows
  - Optimizer estimated cost
  - Number of rows returned
- Database isolation level.

You can also specify escalating levels of information to include in the tracing, as follows:

- *Low level* tracing, which is enabled by default, captures the information shown in the example below. This information includes statement statistics, statement text, and statement iterators.
- *Medium level* tracing captures all of the information included in low-level tracing, plus the list of table names, database name and stored procedure stacks.
- *High level* tracing captures all of the information included in medium-level tracing, plus host variables.

The amount of information traced affects the amount of memory required for this historical data.

You can enable and disable the tracing at any point in time, and you can change the number and size of the trace buffers while the database server is running. If you resize the trace buffer, the database server attempts to maintain the content of the buffer. If the parameters are increased, data will not be truncated. However, if the number or the size of the buffers are reduced, the data in the trace buffers might be truncated or lost.

The number of buffers determines how many SQL statements are traced. Each buffer contains the information for a single SQL statement. By default, an individual trace buffer is a fixed size. If the text information stored in the buffer exceeds the size of the trace buffer, then the data is truncated.

Below is an example of SQL tracing information:

```
select * from syssqltrace where sql_id = 5678;

sql_id            5678
sql_address       4489052648
sql_sid           55
sql_uid           2053
sql_stmttype      6
sql_stmtname      INSERT
sql_finishtime    1140477805
sql_begintxtime   1140477774
sql_runtime       30.86596333400
sql_pgreads       1285
sql_bfreads       19444
sql_rdcache       93.39127751491
sql_bfidxreads    5359
sql_pgwrites      810
sql_bfwrites      17046
sql_wrcache       95.24815205913
sql_lockreq       10603
sql_lockwaits     0
sql_lockwttime    0.00
sql_logspace      60400
sql_sorttotal     0
sql_sortdisk      0
sql_sortmem       0
sql_executions    1
sql_totaltime     30.86596333400
sql_avgtime       30.86596333400
sql_maxtime       30.86596333400
sql_numiowaits    2080
sql_avgiowaits    0.014054286131
sql_totaliowaits  29.23291515300
sql_rowspersec    169.8958799132
sql_estcost       102
sql_estrows       1376
sql_actualrows    5244
sql_sqlerror      0
sql_isamerror     0
sql_isollevel     2
sql_sqlmemory     32608
sql_numiterators  4
sql_database      db3
sql_numtables     3
sql_tablelist     t1
sql_statement     insert into t1 select {+ AVOID_FULL(sysindices) } 0, tabname
```

Figure 32-1. Sample SQL Tracing Information

For an explanation of all table rows, see information on the **syssqltrace** table in the **sysmaster** database section of the *IBM Informix Dynamic Server Administrator's Reference*.

You can enable SQL history tracing globally, so you can view commands that each user ran; or you can enable SQL history tracing for a particular user. You can also disable SQL history tracing globally or for a particular user.

Specifying Startup SQL Tracing Information by Using the SQLTRACE Configuration Parameter

Use the SQLTRACE configuration parameter to control the default tracing behavior when the database server starts. By default, this parameter is not set. The information you set includes the number of SQL statements to trace and the tracing mode.

Prerequisite: Any user who can modify the **$INFORMIXDIR/etc/$ONCONFIG** file can modify the value of the SQLTRACE configuration parameter and effect the startup configuration. However, only user **informix**, **root**, or any DBSA who has been granted connect privileges to the **sysadmin** database can use SQL administration API commands to modify the runtime status of the SQL trace feature.

To specify SQL tracing information when the database server starts:

1. In the **level** field, specify one of the following values:
   - `Low`. This tracing level, which is enabled by default, captures statement statistics, statement text, and statement iterators.
   - `Medium`. This tracing level captures all of the information included in low-level tracing, plus table names, the database name, and stored procedure stacks.
   - `High` tracing. This tracing level captures all of the information included in medium-level tracing, plus host variables.
   - `Off`. This specifies no SQL tracing.
2. In the **ntraces** field, specify the number of SQL statements to trace.
3. In the **size** field, specify the number of kilobytes for the size of the trace buffer. If this buffer size is exceeded, the database server discards saved data.
4. In the **mode** field, specify either:
   - `Global` for all users on the system
   - `User` for users who have tracing enabled by an SQL administration API **task()** function (Specify this if you want to get a sample of the SQL that a small set of users is running.)

For more information on the SQLTRACE configuration parameter, including minimum and maximum values for some fields, see the *IBM Informix Dynamic Server Administrator's Reference*.

Example: The following statement specifies that the database server will gather low-level information on up to 2000 SQL statements executed by all users on the system and will allocate approximately four megabytes of memory (2000 * two kilobytes).

```
SQLTRACE level=LOW,ntraces=2000,size=2,mode=global
```

If you only use a percentage of the allocated buffer space (for example, 42 percent of the buffer space), the amount of memory that is allocated is still two kilobytes.

For more information on SQL tracing, see Chapter 32, "Query Drill-Down," on page 32-1.

If you do not want to set the SQLTRACE configuration parameter to restart the server, you can execute the following SQL administration API command, which provides the same functionality as setting SQLTRACE:

```
execute function task("set sql tracing on", 100,"1k","med","user");
```

After enabling the SQL tracing system in user mode, you can then enable tracing for each user. See "Enabling SQL History Tracing for a Particular User."

For more information on defining **admin()** and **task()** functions, see the *IBM Informix Dynamic Server Administrator's Reference*.

Disabling SQL History Tracing Globally or for a Session

Even if the mode specified in the SQLTRACE configuration parameter is `global` or `user`, you can disable SQL history tracing if you want to completely turn off all user and global tracing and deallocate resources currently in use by the SQL tracing feature. By default, SQL history tracing is off for all users.

To disable global SQL tracing, execute a **task()** or **admin()** function, specifying `set sql tracing off`.

For example, disable global SQL tracing by executing this statement:

```
EXECUTE FUNCTION task('set sql tracing off');
(expression) SQL tracing off.

1 row(s) retrieved.
```

To disable SQL tracing for a particular session, execute a **task()** or **admin()** function, specifying `set sql tracing off` and the session identification number in this format:

```
EXECUTE FUNCTION task("set sql user tracing off",session id);
```

For example, disable SQL tracing for session id 47 by executing this statement:

```
EXECUTE FUNCTION task("set sql user tracing off",47);
```

For more information on defining **admin()** and **task()** functions, see the *IBM Informix Guide to SQL: Syntax*.

Enabling SQL History Tracing for a Particular User

After you specify `user` as the mode in the SQLTRACE configuration parameter, you must execute an SQL administration API **task()** or **admin()** function to turn SQL history tracing on for a particular user.

Global SQL tracing is disabled by default. While global SQL tracing is disabled, you can execute an SQL administration API **task()** or **admin()** function to enable SQL tracing for a particular user.

Prerequisite: Only user **informix**, who by default can connect to the **sysadmin** database, can use SQL administration API commands to modify the runtime status of the SQL trace feature. If user **root** or a member of the DBSA group is granted privileges to connect to the **sysadmin** database, then user root or a member of the DBSA group can also invoke the SQL administration API **task()** and **admin()** functions.

To enable SQL tracing for a particular user, execute a **task()** or **admin()** function, specifying `set sql tracing on`.

For example, enable SQL tracing for a particular user by executing this statement:

```
EXECUTE FUNCTION task("set sql user tracing on", session_id);
```

To enable user low-mode SQL history tracing for all users except **root** or **informix**, you can execute a **task()** or **admin()** function, specifying `set sql tracing on` and information that defines the users.

For example, you can execute the following statement, which enables the tracing of SQL statements of users who are currently connected to the system as long as they are not logged in as user **root** or **informix**.

```
dbaccess sysadmin -<<END
 execute function task("set sql tracing on", 1000, 1,"low","user");
 select task("set sql user tracing on", session_id)
  FROM sysmaster:syssessions
  WHERE username not in ("root","informix");
END
```

For more information on **admin()** and **task()** functions, see the *IBM Informix Dynamic Server Administrator's Reference*.

Enabling Low-Mode SQL History Tracing for All Users

By default, low-mode SQL history tracing is enabled for all users. If you disabled SQL history tracing, you can reset it.

To enable user low mode SQL history tracing:

- Specify `low` as the level and `global` as the mode in the SQLTRACE configuration parameter to enable SQL tracing when the server starts, or
- Execute a **task()** or **admin()** function, specifying `set sql tracing on` to turn SQL tracing on.

For example, you can enable global low-level SQL tracing for all users by executing this statement:

```
EXECUTE FUNCTION task("set sql tracing on", 1000, 1,"low","global");
```

If another a new user logs onto the system after your statement runs, you can enable tracing for the new user. See "Enabling SQL History Tracing for a Particular User" on page 32-5.

For more information on defining **admin()** and **task()** functions, see the *IBM Informix Guide to SQL: Syntax*.

Part 7. Appendixes

Appendix. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix Dynamic Server

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility Features

The following list includes the major accessibility features in IBM Informix Dynamic Server. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Tip: The IBM Informix Dynamic Server Information Center and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard Navigation

This product uses standard Microsoft Windows navigation keys.

Related Accessibility Information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software.

You can view the publications for IBM Informix 4GL in Adobe® Portable Document Format (PDF) using the Adobe Acrobat Reader.

IBM and Accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided ″AS IS″, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Printed in USA

SC23-7748-04

Spine information:

IBM Informix **Version 11.50**

IBM Informix Dynamic Server Administrator's Guide