Administrator's Guide

for Informix® Dynamic Server™

Informix Dynamic Server
Informix Dynamic Server, Developer Edition
Informix Dynamic Server, Workgroup Edition

Version 7.3
February 1998
Part No. 000-4354

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025-1032

Copyright © 1981-1998 by Informix Software, Inc. or its subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, registered in the United States of America as indicated by "®," and in numerous other countries worldwide:

Answers OnLine™; INFORMIX®; Informix®; Illustra™; C-ISAM®; DataBlade®; Dynamic Server™; Gateway™; NewEra™

All other names or marks may be registered trademarks or trademarks of their respective owners.

Documentation Team: Bob Berry, Twila Booth, Lynne Casey, Diana Chase, Bonnie Vaughan

RESTRICTED RIGHTS / SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

List of Chapters

Section I What Is Informix Dynamic Server?

Chapter 1 **Introducing Informix Dynamic Server**

Chapter 2 **Overview of Database Server Administration**

Section II Configuration

Chapter 3 **Configuring the Database Server**

Chapter 4 **Client/Server Communications**

Chapter 5 **What Is Multiple Residency?**

Chapter 6 **Using Multiple Residency**

Section III Modes and Initialization

Chapter 7 **Managing Database Server Operating Modes**

Chapter 8 **Initializing the Database Server**

Section IV Disk, Memory, and Process Management

Chapter 9 **What Is Informix Dynamic Scalable Architecture?**

Chapter 10 **Managing Virtual Processors**

Chapter 11 **Shared Memory**

Chapter 12 **Managing Shared Memory**

Chapter 13 **Where Is Data Stored?**

Chapter 14 **Managing Disk Space**

Chapter 15 **Overview of Table Fragmentation and PDQ**

Section V Logging and Log Administration

Chapter 16 **What Is Logging?**

Chapter 17 **Managing Database-Logging Status**

Chapter 18 **What Is the Logical Log?**

Chapter 19 **Managing Logical-Log Files**

Chapter 20 **What Is Physical Logging?**

Chapter 21 **Managing the Physical Log**

Chapter 22 **What Is Fast Recovery?**

Section VI Fault Tolerance

Chapter 23 **What Is Mirroring?**

Chapter 24 **Using Mirroring**

Chapter 25 **What Is High-Availability Data Replication?**

Chapter 26 **Using High-Availability Data Replication**

Chapter 27 **What Is Consistency Checking?**

Chapter 28 **Situations to Avoid**

Section VII Monitoring

Chapter 29 **Monitoring the Database Server**

Section VIII Distributed Data

Chapter 30 **Multiphase Commit Protocols**

Chapter 31 **Recovering Manually from Failed Two-Phase Commit**

Section IX Reference

Chapter 32 **ON-Monitor for UNIX**

Chapter 33 **Configuration Parameters**

Chapter 34 **The sysmaster Database**

Chapter 35 **Utilities**

Chapter 36 **Message-Log Messages**

Chapter 37 **Interpreting Logical-Log Records**

Chapter 38 **Disk Structures and Storage**

Table of Contents

Introduction

About This Manual 3
- Types of Users 3
- Software Dependencies 4
- Assumptions About Your Locale 4
- Demonstration Database 5

New Features 6

Documentation Conventions 7
- Typographical Conventions 8
- Icon Conventions 8
- Command-Line Conventions 10

Additional Documentation 13
- On-Line Manuals 13
- Printed Manuals 14
- Error Message Files 14
- Documentation Notes, Release Notes, Machine Notes 15

Compliance with Industry Standards 16

Informix Welcomes Your Comments 16

Section I What Is Informix Dynamic Server?

Chapter 1 Introducing Informix Dynamic Server

What Is Informix Dynamic Server? 1-3
- Client/Server Architecture 1-4
- Scalability 1-5
- High Performance 1-6
- Fault Tolerance and High Availability 1-8
- Dynamic System Administration 1-11
- Distributed Data Queries 1-11
- Database Server Security 1-12

Features Beyond the Scope of Dynamic Server 1-12
No Bad-Sector Mapping 1-12
No Scanning or Compression of TEXT and BYTE Data Types 1-13

Chapter 2 Overview of Database Server Administration

Who Is the Database Server Administrator? 2-3
Initial Tasks 2-4
Routine Tasks 2-5
Changing Database Server Operating Modes 2-5
Backing Up Data and Logical-Log Files 2-6
Monitoring Activity 2-6
Checking for Consistency 2-6
Configuration Tasks 2-7
Managing Disk Space 2-7
Managing Database-Logging Status 2-7
Logical-Log Administration 2-7
Physical-Log Administration. 2-8
Using Auditing 2-8
Using Mirroring 2-8
Using Data Replication. 2-9
Managing Shared Memory 2-9
Managing Virtual Processors. 2-10
Managing Parallel Database Query 2-10
Administration Tasks Summary 2-11

Section II Configuration

Chapter 3 Configuring the Database Server

Planning for the Database Server 3-3
Consider Your Priorities 3-3
Consider Your Resources 3-4
Configuring the Operating System 3-5
Operating-System Administration Facilities 3-5
Configuring Operating-System Resources 3-6
Installing the Database Server 3-7
Configuring the Database Server 3-7
Set Environment Variables 3-8
Prepare the ONCONFIG Configuration File 3-10
Allocate Disk Space 3-28

Prepare the Connectivity Information 3-28
Prepare the ON-Archive Configuration File 3-29
Prepare for Global Language Support 3-29
Evaluate Operating-System Parameters 3-29
Start the Database Server and Initialize Disk Space 3-30
Create Dbspaces and Blobspaces 3-30
Perform Administrative Tasks 3-31

Chapter 4 Client/Server Communications

What Is Client/Server Architecture? 4-3
What Is a Network Protocol? 4-4
What Is a Network Programming Interface? 4-5
Windows NT Network Domains 4-5
What Is a Connection? 4-6
What Is a Multiplexed Connection? 4-7
What Connections Does the Database Server Support? 4-8
Network Connections 4-8
Local Connections 4-9
What Interface/Protocol Combinations Are Available on Your Platform? 4-13
What Are Communications Support Services? 4-13
What Is a Communications Support Module? 4-14
How to Configure the DCE-GSS Communications Support Module on UNIX 4-15
Connectivity Files 4-17
Network-Configuration Files 4-17
Network-Security Files 4-22
The $INFORMIXDIR/etc/concsm.cfg file 4-25
The sqlhosts File or Registry 4-27
ONCONFIG Parameters for Connectivity 4-50
The DBSERVERNAME Configuration Parameter 4-51
The DBSERVERALIASES Configuration Parameter 4-51
Environment Variables for Network Connections 4-52
Examples of Client/Server Configurations 4-53
Using a Shared-Memory Connection 4-54
Using a Local Loopback Connection 4-55
Using a Network Connection 4-56
Using Multiple Connection Types 4-57
Accessing Multiple Database Servers 4-59
Using the Relay Module 4-60
Using a Post-6.0 Client Application with a 5.x Database Server 4-63

Chapter 5 What Is Multiple Residency?

Benefits of Multiple Residency 5-3
How Multiple Residency Works 5-4
The Role of the ONCONFIG Environment Variable 5-5
The Role of the SERVERNUM Configuration Parameter . . . 5-5

Chapter 6 Using Multiple Residency

Planning for Multiple Residency 6-3
Preparing for Multiple Residency 6-4
Prepare a Configuration File 6-5
Set the ONCONFIG Environment Variable 6-5
Edit the New Configuration File 6-5
Add Connection Information 6-7
Update the sqlhosts File or Registry 6-7
Initialize Disk Space. 6-7
Prepare Dbspace and Logical-Log Backup Environment . . . 6-8
Update the Operating-System Boot File 6-9
Check Users' INFORMIXSERVER Environment Variables. . . 6-9
Using instmgr.exe 6-10
Adding an Instance of the Database Server 6-10
Deleting an Instance of the Database Server 6-11

Section III Modes and Initialization

Chapter 7 Managing Database Server Operating Modes

Database Server Operating Modes 7-3
Changing Database Server Operating Modes 7-5
From Off-Line to Quiescent 7-5
From Off-Line to On-Line. 7-6
From Quiescent to On-Line 7-6
Gracefully from On-Line to Quiescent 7-7
Immediately from On-Line to Quiescent. 7-7
From Any Mode Immediately to Off-Line 7-8

Chapter 8 Initializing the Database Server

Types of Initialization 8-3
Initialization Commands 8-4
Initialization Steps 8-4
Process Configuration File 8-6
Create Shared-Memory Portions 8-7

Initialize Shared-Memory Structures 8-8
Initialize Disk Space 8-8
Start All Required Virtual Processors 8-8
Make Necessary Conversions 8-9
Initiate Fast Recovery 8-9
Initiate a Checkpoint 8-9
Document Configuration Changes 8-9
Create the oncfg_servername.servernum File 8-10
Drop Temporary Tblspaces 8-10
Set Forced Residency If Specified 8-10
Return Control to User 8-11
Prepare SMI Tables 8-11
After Initialization 8-12

Section IV Disk, Memory, and Process Management

Chapter 9 What Is Informix Dynamic Scalable Architecture?

What Is a Virtual Processor? 9-5
What Is a Thread? 9-5
Types of Virtual Processors 9-7
Advantages of Virtual Processors 9-9
How Virtual Processors Service Threads 9-12
Control Structures 9-13
Context Switching 9-14
Stacks 9-15
Queues 9-16
Multiple Virtual Processors on Windows NT 9-19
Mutexes 9-19
Virtual-Processor Classes 9-20
CPU Virtual Processors 9-20
Disk I/O Virtual Processors 9-24
Network Virtual Processors 9-28
Communications Support Module Virtual Processor 9-35
Optical Virtual Processor 9-35
Audit Virtual Processor 9-36
Miscellaneous Virtual Processor 9-36

Chapter 10 Managing Virtual Processors

Setting Virtual-Processor Configuration Parameters 10-3
Setting Virtual-Processor Configuration Parameters with a Text Editor 10-4
Setting Virtual-Processor Configuration Parameters with ON-Monitor 10-5
Starting and Stopping Virtual Processors 10-7
Adding Virtual Processors in On-Line Mode 10-7
Dropping CPU Virtual Processors in On-Line Mode. 10-10

Chapter 11 Shared Memory

What Is Shared Memory? 11-5
How the Database Server Uses Shared Memory 11-6
How the Database Server Allocates Shared Memory 11-7
How Much Shared Memory Does the Database Server Need? 11-9
What Action Should You Take If SHMTOTAL Is Exceeded? 11-10
What Processes Attach to Shared Memory? 11-11
How a Client Attaches to the Communications Portion . . . 11-11
How Utilities Attach to Shared Memory. 11-12
How Virtual Processors Attach to Shared Memory 11-12
Keeping Shared-Memory Segments Resident 11-17
The Resident Portion of Shared Memory 11-18
Shared-Memory Header 11-19
Shared-Memory Buffer Pool 11-19
The Virtual Portion of Shared Memory 11-25
How the Database Server Manages the Virtual Portion of Shared Memory 11-25
The Virtual Portion of Shared Memory 11-26
The Communications Portion of Shared Memory 11-34
Concurrency Control 11-35
Shared-Memory Mutexes 11-35
Shared-Memory Buffer Locks 11-36
How Database Server Threads Access Shared Buffers 11-37
LRU Queues 11-37
Configuring the Database Server to Read Ahead 11-42
How a Database Server Thread Accesses a Buffer Page. . . . 11-43
How the Database Server Flushes Data to Disk 11-46
Flushing the Physical-Log Buffer 11-47
How the Database Server Synchronizes Buffer Flushing . . . 11-50

How Write Types Describe Flushing Activity 11-51
Flushing the Logical-Log Buffer 11-53
How the Database Server Achieves Data Consistency 11-56
Critical Sections. 11-56
Checkpoints 11-57
Time Stamps 11-60
Buffering TEXT and BYTE Data Types 11-63
Writing TEXT and BYTE Data 11-63

Chapter 12 Managing Shared Memory

Setting Operating-System Shared-Memory Configuration Parameters 12-3
Maximum Operating-System Shared-Memory Segment Size 12-4
Maximum Number of Shared-Memory Identifiers 12-4
Shared-Memory Lower-Boundary Address 12-5
Maximum Amount of Shared Memory for One Process 12-5
Semaphores 12-6
Setting Database Server Shared-Memory Configuration Parameters 12-6
Setting Parameters for Resident Shared Memory with ON-Monitor 12-7
Setting Parameters for Resident Shared Memory with a Text Editor 12-8
Setting Parameters for Virtual Shared Memory with ON-Monitor 12-10
Setting Parameters for Virtual Shared Memory with a Text Editor 12-11
Setting Parameters for Shared-Memory Performance Options with ON-Monitor 12-12
Setting Parameters for Shared-Memory Performance Options with a Text Editor 12-13
Reinitializing Shared Memory 12-14
Turning Residency On or Off for Resident Shared Memory 12-14
Turning Residency On or Off in On-Line Mode 12-14
Turning Residency On or Off for the Next Time You Reinitialize Shared Memory 12-15
Adding a Segment to the Virtual Portion of Shared Memory 12-15
Forcing a Checkpoint 12-16

Chapter 13 Where Is Data Stored?

Overview of Data Storage 13-3
What Are the Physical Units of Storage? 13-5
What Is a Chunk? 13-5
What Is a Page? 13-10
What are Blobpages? 13-11
What Is an Extent? 13-13
What Are the Logical Units of Storage? 13-16
What Is a Dbspace? 13-16
What Is a Blobspace? 13-21
What Is a Database? 13-22
What Is a Table? 13-23
What Is a Temporary Table? 13-24
What Is a Tblspace? 13-27
How Much Disk Space Do You Need to Store Your Data? 13-30
Calculate the Size of the Root Dbspace 13-30
Estimate Space That Databases Require 13-33
Disk-Layout Guidelines 13-33
Dbspace and Chunk Guidelines. 13-34
Table-Location Guidelines 13-36
Sample Disk Layouts 13-40
What Is a Logical-Volume Manager? 13-46

Chapter 14 Managing Disk Space

Allocating Disk Space 14-4
Do You Need to Specify an Offset? 14-5
Allocating a File for Disk Space on UNIX 14-7
Allocating a File for Disk Space on Windows NT 14-8
Allocating Raw Disk Space on UNIX 14-8
Allocating Raw Disk Space on Windows NT 14-9
Initializing Disk Space 14-10
Initializing Disk Space with oninit 14-11
Initializing Disk Space with ON-Monitor 14-11
Creating a Dbspace 14-11
Creating a Temporary Dbspace 14-12
Creating a Dbspace with onspaces 14-13
Creating a Dbspace with ON-Monitor 14-13
Adding a Chunk to a Dbspace 14-14
Adding a Chunk 14-14

Creating a Blobspace 14-16
Determining Database Server Page Size 14-17
Creating a Blobspace with onspaces. 14-17
Creating a Blobspace with ON-Monitor 14-17
Adding a Chunk to a Blobspace 14-18
Dropping a Chunk from a Dbspace with onspaces 14-18
Dropping a Chunk from a Blobspace 14-19
Dropping a Dbspace or Blobspace 14-19
Dropping a Dbspace or Blobspace with onspaces 14-20
Dropping a Dbspace or Blobspace with ON-Monitor 14-20
Optimizing Blobspace Blobpage Size 14-21
Determining Blobspace Storage Efficiency 14-21
Blobspace Storage Statistics. 14-21
Determining Blobpage Fullness with oncheck -pB 14-22

Chapter 15 Overview of Table Fragmentation and PDQ

What Is Fragmentation? 15-3
Fragmentation Goals 15-5
Whose Responsibility Is Fragmentation? 15-6
Fragmentation Strategies 15-6
Using SQL Statements to Perform Fragmentation Tasks 15-9
What Is PDQ? 15-10
High Degree of Parallelism 15-11
When Should You Use PDQ? 15-12
OLTP Applications 15-13
Decision-Support Applications 15-14
Processing Decision-Support Queries 15-15
How Does the Database Server Allocate Resources with PDQ? . . . 15-16
Parameters Used for Controlling PDQ 15-16
How Does the Database Server Use PDQ? 15-18
SQL Operations That Take Advantage of PDQ 15-18
SQL Operations That Do Not Use PDQ 15-21
Update Statistics 15-21
Stored Procedures and Triggers 15-22
Correlated and Uncorrelated Subqueries 15-22
Remote Tables 15-23

Section V Logging and Log Administration

Chapter 16 What Is Logging?

Which Database Server Processes Require Logging? 16-3
What Database Server Activity Is Logged? 16-5
Activity That Is Always Logged 16-6
Activity Logged for Databases with Transaction Logging . . . 16-7
Are Blobs Logged? 16-7
What Is Transaction Logging? 16-8
The Database-Logging Status 16-8
When to Use Transaction Logging 16-10
When to Buffer Transaction Logging 16-11
Who Can Set or Change Logging Status? 16-11

Chapter 17 Managing Database-Logging Status

About Changing Logging Status 17-3
Modifying Database-Logging Status with ON-Archive 17-5
Turning On Transaction Logging with ON-Archive 17-5
Canceling a Logging Operation with ON-Archive 17-6
Ending Logging with ON-Archive 17-6
Changing Buffering Status with ON-Archive 17-7
Making a Database ANSI Compliant with ON-Archive . . . 17-7
Modifying Database-Logging Status with ontape 17-7
Turning On Transaction Logging with ontape 17-7
Ending Logging with ontape. 17-8
Changing Buffering Status with ontape 17-8
Making a Database ANSI Compliant with ontape 17-9
Modifying Database Logging Status with ON-Monitor 17-9

Chapter 18 What Is the Logical Log?

What Is the Logical Log? 18-3
What Is a Logical-Log File? 18-4
How Big Should the Logical Log Be? 18-5
Performance Considerations 18-5
Long-Transaction Consideration 18-6
Logical-Log Size Guidelines 18-6
Determining the Size of the Logical Log 18-7
Preserving Log Space for ON-Archive Tasks 18-8
Enabling the Logs-Full High-Water Mark 18-9
Emergency Log Backup 18-9
What Should Be the Size and Number of Logical-Log Files? . . . 18-11

Where Should Logical-Log Files Be Located? 18-12
How Are Logical-Log Files Identified? 18-12
What Are the Status Flags of Logical-Log Files? 18-13
Point-In-Time Recovery 18-15
Why Do Logical-Log Files Need to Be Backed Up? 18-15
When Are Logical-Log Files Freed? 18-16
When Does the Database Server Attempt to Free a Log File? 18-16
What Happens If the Next Logical-Log File Is Not Free?. . . . 18-16
Avoiding Long Transactions 18-18
What Are the Logical-Log Administration Tasks Required for Blobspaces? 18-21
Switching Logical-Log Files to Activate Blobspaces 18-21
Switching Logical-Log Files to Activate New Blobspace Chunks. 18-22
Backing Up Logical-Log Files to Free Blobpages 18-22
What Is the Logging Process? 18-25
Dbspace Logging 18-25
Blobspace Logging. 18-27

Chapter 19 Managing Logical-Log Files

Adding a Logical-Log File 19-4
Using ON-Monitor to Add a Log File 19-5
Using onparams to Add a Log File 19-5
Adding a Log File with a New Size 19-5
Dropping a Logical-Log File 19-6
Using ON-Monitor to Drop a Logical-Log File 19-7
Using onparams to Drop a Logical-Log File 19-7
Moving a Logical-Log File to Another Dbspace 19-8
An Example of Moving Logical-Log Files 19-8
Changing the Size of Logical-Log Files 19-9
Changing Logical-Log Configuration Parameters 19-10
Changing LOGSIZE or LOGFILES 19-10
Changing LOGSMAX, LTXHWM, or LTXEHWM 19-12
Freeing a Logical-Log File 19-13
Freeing a Log File with Status A 19-13
Freeing a Log File with Status U 19-14
Freeing a Log File with Status U-B 19-14
Freeing a Log File with Status U-C or U-C-L 19-14
Freeing a Log File with Status U-B-L 19-15
Switching to the Next Logical-Log File 19-15

Chapter 20 What Is Physical Logging?

What Is Physical Logging? 20-3
- What Is the Purpose of Physical Logging? 20-4
- What Database Server Activity Is Physically Logged? 20-4

How Big Should the Physical Log Be? 20-5

Where Is the Physical Log Located? 20-8

Details of Physical Logging 20-9
- Page Is Read into the Shared-Memory Buffer Pool 20-9
- A Copy of the Page Buffer Is Stored in the Physical-Log Buffer. 20-10
- Change Is Reflected in the Data Buffer 20-10
- Physical-Log Buffer Is Flushed to the Physical Log 20-10
- Page Buffer Is Flushed 20-11
- When a Checkpoint Occurs 20-11
- How the Physical Log Is Emptied 20-11

Chapter 21 Managing the Physical Log

Changing the Physical-Log Location and Size 21-3
- Why Change Physical-Log Location and Size?. 21-4
- Before You Make the Changes 21-4
- Using ON-Monitor to Change Physical-Log Location or Size 21-5
- Using a Text Editor to Change Physical-Log Location and Size 21-5
- Using onparams to Change Physical-Log Location or Size 21-6

Chapter 22 What Is Fast Recovery?

What Is Fast Recovery? 22-3
- When Is Fast Recovery Needed? 22-4
- When Does the Database Server Initiate Fast Recovery? . . . 22-4
- Fast Recovery and Buffered Logging 22-4
- Fast Recovery and No Logging 22-5

Details of Fast Recovery 22-5
- Returning to the Last-Checkpoint State 22-6
- Finding the Checkpoint Record in the Logical Log 22-6
- Rolling Forward Logical-Log Records 22-7
- Rolling Back Incomplete Transactions 22-8

Section VI Fault Tolerance

Chapter 23 What Is Mirroring?

What Is Mirroring? 23-4
- What Are the Benefits of Mirroring? 23-4
- What Are the Costs of Mirroring? 23-5
- What Happens If You Do Not Mirror? 23-5
- What Should You Mirror? 23-5
- What Mirroring Alternatives Exist? 23-6

The Mirroring Process 23-7
- What Happens When You Create a Mirrored Chunk? 23-7
- What Are Mirror Status Flags? 23-8
- What Is Recovery? 23-8
- What Happens During Processing? 23-9
- What Happens If You Stop Mirroring? 23-11
- What Is the Structure of a Mirrored Chunk? 23-11

Chapter 24 Using Mirroring

Steps Required for Mirroring Data 24-3

Enabling Mirroring 24-4

Allocating Disk Space for Mirrored Data 24-5

Starting Mirroring 24-6
- Mirroring the Root Dbspace During Initialization 24-7
- Starting Mirroring for Unmirrored Dbspaces 24-8
- Starting Mirroring for New Dbspaces 24-8

Adding Mirrored Chunks 24-9

Changing the Mirror Status 24-10
- Taking Down a Mirrored Chunk 24-10
- Recovering a Mirrored Chunk 24-11

Relinking a Chunk to a Device After a Disk Failure 24-12

Ending Mirroring 24-12
- Ending Mirroring with ON-Monitor 24-13
- Ending Mirroring with onspaces 24-13

Chapter 25 What Is High-Availability Data Replication?

What Is Data Replication? 25-4
What Is High-Availability Data Replication? 25-4
How Does High-Availability Data Replication Work? 25-8
How Does Data Initially Replicate? 25-8
Reproducing Updates to the Primary Database Server 25-9
What Threads Handle High-Availability Data Replication? . . 25-13
Checkpoints Between Database Servers 25-14
How Is Data Synchronization Tracked? 25-14
HIgh-Availability Data-Replication Failures 25-15
What Are High-Availability Data-Replication Failures? . . . 25-15
How Are High-Availability Data-Replication Failures Detected? 25-16
What Happens When a High-Availability Data-Replication Failure Is Detected? 25-16
Considerations After High-Availability Data-Replication Failure 25-17
Redirection and Connectivity for Data-Replication Clients 25-22
Designing Clients for Redirection 25-23
Automatic Redirection with DBPATH 25-23
Administrator-Controlled Redirection with the Connectivity Information. 25-25
User-Controlled Redirection with INFORMIXSERVER. . . . 25-29
Handling Redirection Within an Application 25-30
Comparison of Different Redirection Mechanisms 25-33
Designing High-Availability Data-Replication Clients 25-34
Setting Lock Mode to Wait for Access to Primary Database Server 25-34
Designing Clients to Use the Secondary Database Server . . . 25-35

Chapter 26 Using High-Availability Data Replication

Planning for High-Availability Data Replication 26-4
Configuring High-Availability Data Replication 26-5
Meeting Hardware and Operating-System Requirements . . . 26-5
Meeting Database and Data Requirements 26-6
Meeting Database Server Configuration Requirements. . . . 26-6
Configuring High-Availability Data-Replication Connectivity 26-9
Starting High-Availability Data Replication for the First Time . . . 26-11

Performing Basic Administration Tasks 26-15
Changing Database Server Configuration Parameters 26-15
Dbspace and Logical-Log File Backups. 26-16
Changing the Logging Status of Databases 26-16
Adding and Dropping Chunks, Dbspaces, and Blobspaces 26-17
Using and Changing Mirroring of Chunks 26-17
Managing the Physical Log. 26-18
Managing the Logical Log 26-19
Managing Virtual Processors 26-19
Managing Shared Memory 26-19
Changing the Database Server Mode 26-20
Changing the Database Server Type 26-21
Changing the Type of the Primary Database Server 26-22
Changing the Type of the Secondary Database Server 26-22
Restoring Data If Media Failure Occurs 26-23
Restoring After Media Failure on the Primary Database Server 26-23
Restoring After Media Failure on the Secondary Database Server 26-24
Restarting High-Availability Data Replication After a Failure . . . 26-26
Restarting After Critical Data Is Damaged 26-26
Restarting If Critical Data Is Not Damaged 26-28

Chapter 27 What Is Consistency Checking?

Performing Periodic Consistency Checking 27-4
Verify Consistency. 27-4
Monitor for Data Inconsistency 27-7
Retain Consistent Level-0 Dbspace 27-8
Dealing with Corruption 27-9
Symptoms of Corruption 27-9
Run oncheck First 27-9
I/O Errors on a Chunk 27-10
Collecting Diagnostic Information 27-11

Chapter 28 Situations to Avoid

Situations to Avoid in Administering the Database Server 28-3

Section VII Monitoring

Chapter 29 Monitoring the Database Server

Information That You Can Monitor 29-5
Sources of Information for Monitoring the Database Server . . . 29-6
What Is the Message Log?. 29-7
Event Alarm 29-8
What Is the Console? 29-12
Monitoring with ON-Monitor 29-12
Monitoring with SMI Tables 29-12
Monitoring with onstat and oncheck Utilities 29-12
Monitoring with onperf 29-13
Monitoring with the onstat Banner Line 29-13
Monitoring Configuration Information 29-14
Monitoring Checkpoint Information 29-16
Monitoring Shared Memory 29-18
Monitoring Shared-Memory Segments 29-18
Monitoring Shared-Memory Profile 29-19
Monitoring Buffers 29-20
Monitoring Buffer-Pool Activity 29-23
Monitoring Latches 29-27
Monitoring Locks 29-29
Monitoring Active Tblspaces 29-32
Monitoring Virtual Processors 29-33
Monitoring Sessions and Threads 29-35
Monitoring PDQ Resources and Queries 29-40
Monitoring Transactions 29-41
Monitoring Databases 29-43
Monitoring Logging Activity 29-44
Monitoring Logical-Log Files 29-44
Monitoring the Physical-Log File 29-47
Monitoring the Physical-Log and Logical-Log Buffers 29-49
Monitoring the Database Server for Disabling I/O Errors 29-52
Using the Message Log to Monitor Disabling I/O Errors . . . 29-52
Using Event Alarms to Monitor Disabling I/O Errors 29-53
Monitoring Disk Usage 29-53
Monitoring Chunks 29-53
Monitoring Tblspaces and Extents 29-60
Monitoring TEXT and BYTE Data in a Blobspace 29-63
Monitoring TEXT and BYTE Data in a Dbspace 29-69
Monitoring High-Availability Data-Replication Status 29-71

Section VIII Distributed Data

Chapter 30 Multiphase Commit Protocols

Two-Phase Commit Protocol 30-3
When Is the Two-Phase Commit Protocol Used? 30-3
What Goals Does the Two-Phase Commit Protocol Achieve? 30-4
Two-Phase Commit Concepts 30-5
Phases of the Two-Phase Commit Protocol 30-6
Examples of Two-Phase Commit Transactions 30-8
How the Two-Phase Commit Protocol Handles Failures 30-10
Presumed-Abort Optimization 30-17
Independent Actions 30-18
What Initiates Independent Action? 30-19
Possible Results of Independent Action 30-19
The Heuristic Rollback Scenario 30-22
The Heuristic End-Transaction Scenario 30-26
Tracking a Global Transaction 30-28
Two-Phase Commit Protocol Errors 30-29
Two-Phase Commit and Logical-Log Records 30-30
Logical-Log Records When the Transaction Commits 30-31
Logical-Log Records Written During a Heuristic Rollback . . . 30-33
Logical-Log Records Written After a Heuristic End Transaction 30-35
Configuration Parameters Used in Two-Phase Commits 30-37
Function of the DEADLOCK_TIMEOUT Parameter 30-37
Function of the TXTIMEOUT Parameter 30-37
Heterogeneous Commit Protocol 30-38
Which Gateways Can Participate in a Heterogeneous Commit Transaction? 30-39
Enabling and Disabling Heterogeneous Commit 30-40
How Does Heterogeneous Commit Work 30-41
Implications of a Failed Heterogeneous Commit 30-43

Chapter 31 Recovering Manually from Failed Two-Phase Commit

Procedure to Determine If Manual Recovery Is Required 31-3
Determine If a Transaction Was Implemented Inconsistently 31-4
Determine If the Distributed Database Contains Inconsistent Data 31-6
Decide If Action Is Needed to Correct the Situation 31-9
Example of Manual Recovery 31-10

Section IX Reference

Chapter 32 ON-Monitor for UNIX

Using ON-Monitor 32-3
Help and Navigation Within ON-Monitor 32-4
Executing Shell Commands from Within ON-Monitor 32-4
ON-Monitor Screen Options 32-4
Setting Configuration Parameters with ON-Monitor 32-12

Chapter 33 Configuration Parameters

ONCONFIG File Conventions 33-5
ONCONFIG Parameters 33-6
ADTERR, ADTMODE, ADTPATH, and ADTSIZE 33-8
AFF_NPROCS 33-9
AFF_SPROC 33-10
ALARMPROGRAM 33-11
BAR_ACT_LOG, BAR_BSALIB_PATH, BAR_MAX_BACKUP, BAR_NB_XPORT_COUNT, BAR_RETRY, and BAR_XFER_BUF_SIZE 33-11
BUFFERS 33-12
CKPTINTVL 33-13
CLEANERS 33-14
CONSOLE 33-15
DATASKIP 33-16
DBSERVERALIASES 33-17
DBSERVERNAME 33-18
DBSPACETEMP 33-19
DEADLOCK_TIMEOUT 33-21
DRAUTO 33-22
DRINTERVAL 33-23
DRLOSTFOUND 33-24
DRTIMEOUT 33-25
DS_MAX_QUERIES 33-26
DS_MAX_SCANS 33-27
DS_TOTAL_MEMORY 33-28
DUMPCNT 33-32
DUMPCORE 33-33
DUMPDIR 33-34
DUMPGCORE 33-34

DUMPSHMEM 33-35
FILLFACTOR 33-36
HETERO_COMMIT 33-36
LBU_PRESERVE 33-37
LOCKS 33-38
LOG_BACKUP_MODE 33-39
LOGBUFF 33-40
LOGFILES 33-41
LOGSIZE 33-42
LOGSMAX 33-43
LRUS 33-44
LRU_MAX_DIRTY 33-45
LRU_MIN_DIRTY 33-45
LTAPEBLK 33-46
LTAPEDEV 33-47
LTAPESIZE 33-48
LTXEHWM 33-49
LTXHWM 33-50
MAX_PDQPRIORITY 33-51
MIRROR 33-52
MIRROROFFSET 33-53
MIRRORPATH 33-53
MSGPATH 33-54
MULTIPROCESSOR 33-55
NETTYPE 33-56
NOAGE 33-60
NUMAIOVPS 33-61
NUMCPUVPS 33-62
OFF_RECVRY_THREADS 33-63
ON_RECVRY_THREADS 33-64
ONDBSPACEDOWN 33-65
OPCACHEMAX 33-66
OPTCOMPIND 33-67
PHYSBUFF 33-68
PHYSDBS 33-69
PHYSFILE 33-70
RA_PAGES 33-71
RA_THRESHOLD 33-72
RESIDENT 33-73

RESTARTABLE_RESTORE 33-74
ROOTNAME 33-75
ROOTOFFSET 33-75
ROOTPATH 33-76
ROOTSIZE 33-77
SERVERNUM 33-78
SHMADD 33-79
SHMBASE 33-80
SHMTOTAL 33-81
SHMVIRTSIZE 33-82
SINGLE_CPU_VP 33-83
STACKSIZE 33-84
STAGEBLOB 33-85
TAPEBLK 33-86
TAPEDEV 33-87
TAPESIZE 33-89
TXTIMEOUT 33-90
USEOSTIME 33-91

Chapter 34 The sysmaster Database

What Is the sysmaster Database? 34-3
Using the System-Monitoring Interface 34-5
What Are the SMI Tables? 34-5
Accessing SMI Tables 34-6
The System-Monitoring Interface Tables 34-8
sysadtinfo 34-9
sysaudit 34-10
syschkio 34-11
syschunks 34-11
sysconfig 34-13
sysdatabases 34-14
sysdbslocale 34-15
sysdbspaces 34-15
sysdri 34-17
sysextents 34-18
syslocks 34-18
syslogs 34-19

sysprofile 34-20
sysptprof 34-23
syssesprof. 34-24
syssessions 34-26
sysseswts 34-28
systabnames 34-29
sysvpprof 34-30
The SMI Tables Map 34-30
Using SMI Tables to Obtain onstat Information 34-33

Chapter 35 Utilities

The -V Option 35-4
Multibyte Characters 35-4
oncheck: Check, Repair, or Display 35-5
ondblog: Change Logging Mode 35-18
oninit: Initialize Dynamic Server 35-20
onlog: Display Logical-Log Contents 35-23
onmode: Mode and Shared-Memory Changes 35-28
onparams: Modify Log-Configuration Parameters 35-43
onspaces: Manage Database Spaces 35-48
onstat: Monitor Database Server Operation 35-62
ontape: Logging, Archives, and Restore 35-95

Chapter 36 Message-Log Messages

How the Messages Are Ordered in This Chapter 36-3
Message Categories 36-4
Messages: A-B 36-4
Messages: C 36-6
Messages: D-E-F 36-14
Messages: G-H-I 36-19
Messages: J-K-L-M 36-20
Messages: N-O-P 36-26
Messages: Q-R-S 36-33
Messages: T-U-V 36-38
Messages: W-X-Y-Z 36-42
Messages: Symbols 36-43

Chapter 37 Interpreting Logical-Log Records

Reading Logical-Log Records 37-3
Transactions That Drop a Table or Index 37-4
Transactions That Are Rolled Back 37-4
Checkpoints with Active Transactions 37-5
Distributed Transactions 37-5
Logical-Log Record Structure 37-6
Logical-Log Record Header 37-6
Logical-Log Record Types and Additional Columns 37-7

Chapter 38 Disk Structures and Storage

Dbspace Structure and Storage 38-4
Structure of the Root Dbspace 38-4
Structure of a Regular Dbspace 38-15
Structure of a Mirrored Chunk 38-17
Structure of the Chunk Free-List Page 38-18
Structure of the Tblspace Tblspace 38-19
Structure of the Database Tblspace 38-24
Structure of a Dbspace Bit-Map Page 38-25
Structure and Allocation of an Extent 38-27
Structure and Storage of a Dbspace Page 38-34
Structure of B-Tree Index Pages 38-48
Blobspace Structure and Storage 38-59
Structure of a Blobspace 38-59
TEXT and BYTE Data Storage and the Descriptor 38-61
Structure of a Dbspace Blobpage 38-62
Blobspace Page Types 38-64
Structure of a Blobspace Blobpage 38-65
Database and Table Creation: What Happens on Disk 38-68
Creating a Database 38-68
Creating a Table 38-69

Appendix A Files That the Database Server Uses

Appendix B Trapping Errors

Index

Introduction

About This Manual 3
Types of Users 3
Software Dependencies 4
Assumptions About Your Locale 4
Demonstration Database 5

New Features 6

Documentation Conventions 7
Typographical Conventions 8
Icon Conventions 8
Comment Icons 9
Feature, Product, and Platform Icons 9
Command-Line Conventions 10
How to Read a Command-Line Diagram 12

Additional Documentation 13
On-Line Manuals 13
Printed Manuals 14
Error Message Files 14
Documentation Notes, Release Notes, Machine Notes 15

Compliance with Industry Standards 16

Informix Welcomes Your Comments 16

Read this introduction for an overview of the information provided in this manual and for an understanding of the documentation conventions used.

About This Manual

This manual is a complete guide to the features that make up Informix Dynamic Server. This book is both a user guide and a reference manual. The first eight sections cover important basic information about the product. The last section contains reference material for using Dynamic Server.

This manual is intended to help you understand, install, configure, and use Dynamic Server to meet your needs.

Types of Users

This manual is for the following users:

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

This manual assumes 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 database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to *Getting Started with Informix Dynamic Server* for a list of supplementary titles.

Software Dependencies

This manual assumes that your database server is one of the following products:

- Informix Dynamic Server, Version 7.3
- Informix Dynamic Server, Developer Edition, Version 7.3.
- Informix Dynamic Server, Workgroup Edition, Version 7.3.

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a GLS (Global Language Support) locale.

This manual assumes that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for dates, times, 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(s). For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

Demonstration Database

The DB-Access utility, which is provided with your Informix database server products, includes a demonstration database called **stores7** that contains information about a fictitious wholesale sporting-goods distributor. You can use SQL scripts provided with DB-Access to derive a second database, called **sales_demo**. This database illustrates a dimensional schema for data-warehousing applications. Sample command files are also included for creating and populating these databases.

Many examples in Informix manuals are based on the **stores7** demonstration database. The **stores7** database is described in detail and its contents are listed in the *Informix Guide to SQL: Reference*.

The scripts that you use to install the demonstration databases reside in the **$INFORMIXDIR/bin** directory on UNIX platforms and the **%INFORMIXDIR%\bin** directory on Windows NT platforms. For a complete explanation of how to create and populate the **stores7** demonstration database, refer to the *DB-Access User Manual*. For an explanation of how to create and populate the **sales_demo** database, refer to the *Informix Guide to Database Design and Implementation*.

New Features

Most of the new features for Version 7.3 of Informix Dynamic Server fall into five major areas:

- Performance
- Reliability, availability, and serviceability
- Manageability
- Windows NT-specific features
- Application migration

Several additional features affect connectivity, replication, and the optical subsystem. For a comprehensive list of new features, see the release notes for the database server.

This manual includes information about the following new features:

- Manageability
  - Informix Enterprise Command Center provides the ability to manage your database environment on both UNIX and Windows NT from a single Windows NT console.
  - On UNIX, you can interface to the DCE-GSS communication support module to provide user authentication, message integrity and message privacy.
- Reliability, availability, and serviceability
  - New RESTARTABLE_RESTORE configuration parameter enables the database server, with ON-Bar, to perform a restart of a failed restore.
  - New **oncheck** options, **-R** and **w**, provide flexibility, the option of faster execution, and greater access to tables.
- Performance

  Using the SET (TABLE or INDEX) statement, users can request that one or more fragments of a table or index remain resident in shared memory, thereby reducing disk I/O and improving access time for the table.

- Windows NT-specific features
  - You can now create and run multiple instances of the database server on Windows NT.
  - You can now allocate raw disk space, in addition to NTFS file space, for database server disk space.
  - You can now store TEXT and BYTE data on an optical subsystem.
- Multiplexed Connections

  You can configure the database server to use shared connection threads so that the connection threads are able to service multiple database connections over a single network connection.
- Optical Subsystem

  Informix Dynamic Server now provides shared library support for the Informix Optical Subsystem; you no longer need to order the Optical Subsystem as a separate product.
- Enterprise Replication

  The **g** option in the **sqlhosts** file or registry enables you to group multiple database server entries as one logical entity. Enterprise Replication uses groups to declare database servers for replication.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other Informix manuals.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Command-line conventions

Typographical Conventions

This manual uses the following standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
italics	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
`monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
♦	This symbol indicates the end of feature-, product-, platform-, or compliance-specific information within a table or section.
→	This symbol indicates a menu item. For example, "Choose **Tools→Options**" means choose the **Options** item from the **Tools** menu.

Tip: *When you are instructed to "enter" characters or to "execute" a command, immediately press* RETURN *after you type the indicated information on your keyboard. When you are instructed to "type" the text or to "press" other keys, you do not need to press* RETURN.

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

Comment Icons

Comment icons identify warnings, important notes, or tips. This information is always displayed in italics.

Icon	Description
	The *warning* icon identifies vital instructions, cautions, or critical information.
	The *important* icon identifies significant information about the feature or operation that is being described.
	The *tip* icon identifies additional details or shortcuts for the functionality that is being described.

Feature, Product, and Platform Icons

Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
GLS	Identifies information that relates to the Informix GLS feature.
IDS	Identifies information that is specific to Dynamic Server and its editions. However, in some cases, the identified section applies only to Informix Dynamic Server and not to Informix Dynamic Server, Workgroup and Developer Editions. Such information is clearly identified.

(1 of 2)

Icon	Description
UNIX	Identifies information that is specific to the UNIX platform.
W/D	Identifies information that is specific to Informix Dynamic Server, Workgroup and Developer Editions.
WIN NT	Identifies information that is specific to the Windows NT environment.

(2 of 2)

These icons can apply to a row in a table, one or more paragraphs, or an entire section. If an icon appears next to a section heading, the information that applies to the indicated feature, product, or platform ends at the next heading at the same or higher level. A ♦ symbol indicates the end of the feature-, product-, or platform-specific information that appears within a table or a set of paragraphs within a section.

Command-Line Conventions

This section defines and illustrates the format of commands that are available in Informix products. These commands have their own conventions, which might include alternative forms of a command, required and optional parts of the command, and so forth.

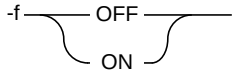
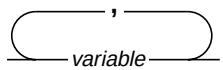
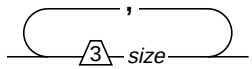
Dynamic Server supports a variety of command-line options. Each valid command-line option is illustrated in a diagram in Chapter 35, "Utilities."

Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper-left corner with a command. It ends at the upper-right corner with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes a valid form of the command. You must supply a value for words that are in italics.

You might encounter one or more of the following elements on a command-line path.

Element	Description
command	This required element is usually the product name or other short word that invokes the product or calls the compiler or preprocessor script for a compiled Informix product. It might appear alone or precede one or more options. You must spell a command exactly as shown and use lowercase letters.
variable	A word in italics represents a value that you must supply, such as a database, file, or program name. A table following the diagram explains the value.
-flag	A flag is usually an abbreviation for a function, menu, or option name or for a compiler or preprocessor argument. You must enter a flag exactly as shown, including the preceding hyphen.
.ext	A filename extension, such as **.sql** or **.cob**, might follow a variable that represents a filename. Type this extension exactly as shown, immediately after the name of the file. The extension might be optional in certain products.
(. , ; + * - /)	Punctuation and mathematical notations are literal symbols that you must enter exactly as shown.
' '	Single quotes are literal symbols that you must enter as shown.
Privileges p. 5-17 Privileges	A reference in a box represents a subdiagram. Imagine that the subdiagram is spliced into the main diagram at this point. When a page number is not specified, the subdiagram appears on the same page.
Change Database Format with onmode -b *see* IMG	A reference to IMG in this manual refers to the *Informix Migration Guide*. Imagine that the subdiagram is spliced into the main diagram at this point.
Archive database server *see* ABG	A reference to ABG in this manual refer to the *Archive and Backup Guide for Informix Dynamic Server*. Imagine that the subdiagram is spliced into the main diagram at this point.

(1 of 2)

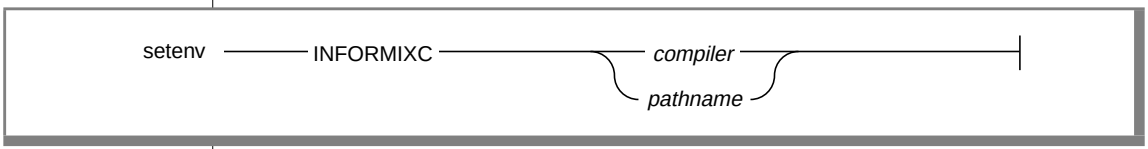
Element	Description
	A shaded option is the default action.
	Syntax within a pair of arrows indicates a subdiagram.
	The vertical line terminates the command.
	A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)
	A loop indicates a path that you can repeat. Punctuation along the top of the loop indicates the separator symbol for list items.
	A gate (3) on a path indicates that you can only use that path the indicated number of times, even if it is part of a larger loop. Here you can specify *size* no more than three times within this statement segment.

(2 of 2)

How to Read a Command-Line Diagram

Figure 1 shows a command-line diagram that uses some of the elements that are listed in the previous table.

Figure 1
Example of a Command-Line Diagram



To construct a command correctly, start at the top left with the command. Then follow the diagram to the right, including the elements that you want. The elements in the diagram are case sensitive.

Figure 1 diagrams the following steps:

1. Type the word `setenv`.
2. Type the word `INFORMIXC`.
3. Supply either a compiler name or pathname.

   After you choose *compiler* or *pathname*, you come to the terminator. Your command is complete.
4. Press RETURN to execute the command.

Additional Documentation

For additional information, you might want to refer to the following types of documentation:

- On-line manuals
- Printed manuals
- Error message files
- Documentation notes, release notes, and machine notes

On-Line Manuals

An Answers OnLine CD that contains Informix manuals in electronic format is provided with your Informix products. You can install the documentation or access it directly from the CD. For information about how to install, read, and print on-line manuals, see the installation insert that accompanies Answers OnLine.

Printed Manuals

To order printed manuals, call 1-800-331-1763 or send email to moreinfo@informix.com. Please provide the following information when you place your order:

- The documentation that you need
- The quantity that you need
- Your name, address, and telephone number

Error Message Files

Informix software products provide ASCII files that contain all of the Informix error messages and their corrective actions. For a detailed description of these error messages, refer to *Informix Error Messages* in Answers OnLine.

UNIX

To read the error messages under UNIX, you can use the following commands.

Command	Description
finderr	Displays error messages on line
rofferr	Formats error messages for printing

♦

WIN NT

To read error messages and corrective actions under Windows NT, use the **Informix Find Error** utility. To display this utility, choose **Start→Programs→Informix** from the Task Bar. ♦

Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following sections describe the on-line files that supplement the information in this manual. Please examine these files before you begin using your database server. They contain vital information about application and performance issues.

UNIX

On UNIX platforms, the following on-line files appear in the **$INFORMIXDIR/release/en_us/0333** directory.

On-Line File	Purpose
ADMINDOC_7.3	The documentation-notes file for your version of this manual describes features that are not covered in this manual or that have been modified since publication.
SERVERS_7.3	The release-notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
IDS_7.3	The machine-notes file describes any special actions that are required to configure and use Informix products on your computer. Machine Notes are named for the product described.

♦

WIN NT

The following items appear in the Informix folder. To display this folder, choose **Start→Programs→Informix** from the Task Bar.

Item	Description
Documentation Notes	This item includes additions or corrections to manuals, along with information about features that may not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

Machine notes do not apply to Windows NT platforms. ♦

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. 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 Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
SCT Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025

If you prefer to send email, our address is:

doc@informix.com

Or send a facsimile to the Informix Technical Publications Department at:

650-926-6571

We appreciate your feedback.

Section I

What Is Informix Dynamic Server?

Chapter

1

Introducing Informix Dynamic Server

What Is Informix Dynamic Server? 1-3
Client / Server Architecture 1-4
Client Application Types 1-4
Connecting Clients and Database Servers 1-5
Scalability. 1-5
High Performance 1-6
Unbuffered Disk Management 1-6
Dynamic Shared-Memory Management 1-7
Dynamic Thread Allocation 1-7
Fragmentation and Parallelization 1-8
Fault Tolerance and High Availability 1-8
Dbspace and Logical-Log Backups of Transaction Records 1-9
Fast Recovery 1-9
Mirroring 1-10
High-Availability Data Replication. 1-10
Point-in-Time Recovery 1-10
Dynamic System Administration. 1-11
Distributed Data Queries 1-11
Database Server Security 1-12

Features Beyond the Scope of Dynamic Server 1-12
No Bad-Sector Mapping. 1-12
No Scanning or Compression of TEXT and BYTE Data Types . . . 1-13

This chapter introduces Informix Dynamic Server. This chapter includes the following information:

- What is Informix Dynamic Server?
- Dynamic Server features
- Features beyond the scope of Dynamic Server

These sections provide an overview of the database server and direct you to more detailed information in this document or in other documents.

What Is Informix Dynamic Server?

Informix Dynamic Server is a multithreaded *relational database* server that exploits symmetric multiprocessor (SMP) and uniprocessor architectures.

Dynamic Server provides the following features:

- Client / server architecture
- Scalability
- High performance
- Fault tolerance and high availability
- Dynamic system administration
- Distributed data queries
- Database server security

Each of these features is explained in the following sections.

Client/Server Architecture

Dynamic Server is a *database server* that processes requests for data from client applications. The *client* is an application program that you run to request information from a database.

The database server accesses the requested information from its databases and sends back the results to the client applications. Accessing the database includes activities such as coordinating concurrent requests from multiple clients, performing read and write operations to the databases, and enforcing physical and logical consistency on the data.

Client applications use Structured Query Language (SQL) to send requests for data to the database server. Client programs include the DB-Access utility and programs that you write using an Informix API such as INFORMIX-ESQL/C or INFORMIX-CLI.

Client Application Types

There are two major classes of applications that operate on data that is in a relational database:

- On-line transaction processing (OLTP) applications
- Decision-support system (DSS) applications

OLTP Applications

OLTP applications are often used to capture new data or update existing data. These operations typically involve quick, indexed access to a small number of rows. An order-entry system is a typical example of an OLTP application. OLTP applications are often multiuser applications with acceptable response times measures in fractions of seconds. OLTP applications have the following characteristics:

- Simple transactions that involve small amounts of data
- Indexed access to data
- Many users
- Frequent requests
- Fast response times

DSS Applications

DSS applications are often used to report on or consolidate data that has been captured through OLTP operations over time. These applications provide information that is often used for accounting, strategic planning, and decision-making. Data within the database is typically queried but not updated during DSS operations. Typical DSS applications include payroll, inventory, and financial reports.

Connecting Clients and Database Servers

A client application communicates with the database server through the connection facilities that the database server provides.

At the source-code level, a client connects to the database server through an SQL statement. Beyond that, the client's use of connection facilities is transparent to the application.

As the database administrator, you specify the types of connections that the database server supports in a connectivity-information file called **sqlhosts**. The **sqlhosts** file contains the names of each of the database servers, and any aliases, to which the clients on a computer can connect. The connections facilities as well as **sqlhosts** are fully described in Chapter 4, "Client / Server Communications."

Scalability

Informix *dynamic scalable architecture* (DSA) describes the capability of an Informix database server to scale its resources to the demands that applications place on it. A key element of DSA is the virtual processors that manage central processing, disk I/O, networking, and optical functions in parallel.

The dynamic-scalable architecture is described in Chapter 9, "What Is Informix Dynamic Scalable Architecture?" To understand how the database server manages shared memory to scale performance, see Chapter 11, "Shared Memory." For tuning and performance information, refer to your *Performance Guide*.

High Performance

Dynamic Server achieves high performance through the following mechanisms:

- Raw disk management
- Dynamic shared-memory management
- Dynamic thread allocation
- Parallelization

Each of these mechanisms is explained in the following paragraphs.

Unbuffered Disk Management

The database server uses unbuffered disk access to improve the speed of I/O operations. UNIX platforms provide unbuffered access with character-special devices (also known as *raw* disk devices). Windows NT platforms provide unbuffered access through both unbuffered files and raw disk devices. For more information about character-special devices, refer to your UNIX operating-system documentation. For more information about unbuffered files on Windows NT platforms, refer to your Windows NT operating-system documentation.

When you store tables on raw disks or unbuffered files, the database server can manage the physical organization of data and minimize disk I/O. When you store tables in this manner, you can receive the following performance advantages:

- The database server optimizes table access by guaranteeing that rows are stored contiguously.
- The database server bypasses operating-system I/O overhead by performing direct data transfers between disk and shared memory.

UNIX

On UNIX, if performance is not a primary concern, you can configure the database server to stored data in regular (buffered) operating-system files (which are also known as *cooked* files). When the database server uses buffered filed, the database server manages the file contents, but the operating system manages the disk I/O. ♦

WIN NT

In the Windows NT environment, regular buffered files correspond to cooked files on UNIX. However, the database server uses unbuffered files rather than buffered files. Informix recommends that you use unbuffered files to simplify disk administration. ♦

For more information about how the database server uses disk space, see Chapter 13, "Where Is Data Stored?"

Dynamic Shared-Memory Management

All applications that use a single instance of a database server share data in the memory space of the database server. After one application reads data from a table, other applications can access whatever data is already in memory. Through shared-memory management, the database server minimizes disk access and the associated impact on performance.

Shared memory contains both data from the database and control information. Because the data needed by various applications is located in a single, shared portion of memory, all control information needed to manage access to that data can be located in the same place. Dynamic Server adds memory dynamically as it needs it, and you, as the administrator, can also add segments to shared memory if necessary. For information on how to add a segment to shared memory, refer to Chapter 12, "Managing Shared Memory."

Dynamic Thread Allocation

The database server supports multiple client applications using a relatively small number of processes called virtual processors. A virtual processor is a multithreaded process that can serve multiple clients and, where necessary, run multiple threads to work in parallel for a single query. In this way, the database server provides a flexible architecture that provides dynamic load balancing for both on-line transaction processing (OLTP) and for decision-support applications.

WIN NT/95

On Windows NT platforms, a virtual processor is implemented as a Windows NT thread rather than as a process. Therefore, on Windows NT both the operating system and the database server are multithreaded such that database server threads run within Windows NT threads.

Important: *To keep the distinction clear between database server threads and Windows NT threads, this manual uses the term process to refer to both a UNIX process and a Windows NT thread.* ♦

For a description of the Dynamic Server dynamic scalable architecture, refer to Chapter 9, "What Is Informix Dynamic Scalable Architecture?"

Fragmentation and Parallelization

The database server uses local table partitioning (also called *fragmentation*) to intelligently distribute tables across disks to improve performance. If you have very large databases (VLDBs), the ability to fragment data is important if you want to efficiently manage the data.

The database server can allocate multiple threads to work in parallel on a single query. This feature is known as the parallel database query (PDQ) feature.

The PDQ feature is most effective when you use it with the fragmentation feature. For an overview of fragmentation and PDQ, refer to Chapter 15, "Overview of Table Fragmentation and PDQ."

Fault Tolerance and High Availability

The database server uses the following logging and recovery mechanisms to protect data integrity and consistency in the event of an operating-system or media failure:

- Dbspace and logical-log backups of transaction records
- Fast recovery
- Mirroring
- Data replication
- Point-in-time recovery

Dbspace and Logical-Log Backups of Transaction Records

Dynamic Server provides you with the ability to back up the data that it manages and also store changes to the database server and data since the backup was performed. The changes are stored in *logical-log files*.

As explained in the *Backup and Restore Guide* (for ON-Bar) and in the *Archive and Backup Guide* (for ON-Archive and **ontape**), the database server allows you to create backup tapes and logical-log backup tapes while users are accessing databases. You can also use on-line archiving to create incremental backups. Incremental backups enable you to back up only data that has changed since the last backup, which reduces the amount of time that a backup would otherwise require.

After a media failure, if critical data was not damaged (and Dynamic Server remains in on-line mode), you can restore only the data that was on the failed media, leaving other data available during the restore.

Fast Recovery

When the database server starts up, it checks if 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. During fast recovery, the database server uses its *logical log* and *physical log* to perform the following operations:

- Restore the databases to their state at the last checkpoint
- Roll forward all committed transactions since the last checkpoint
- Roll back any uncommitted transactions

The database server spawns multiple threads to work in parallel during fast recovery. Fast recovery is explained in detail in Chapter 22, "What Is Fast Recovery?"

Mirroring

When you use disk mirroring, the database server writes data to two locations. Mirroring eliminates data losses due to media (hardware) crashes. If mirrored data becomes unavailable for any reason, the mirror of the data is accessed immediately and transparently to users. Mirroring is explained in Chapter 23, "What Is Mirroring?"

High-Availability Data Replication

If your organization requires a high degree of availability, you can replicate the database server and its databases, running simultaneously on a second computer. Replicating the database server and its databases provides you with a backup system in the event of a catastrophic failure; if one site experiences a disaster, applications can be directed immediately to use the second database server in the pair. For more information about data replication, refer to Chapter 25, "What Is High-Availability Data Replication?"

Running data replication also allows you to balance read-only applications (for example, decision-support applications) across both database servers in the data-replication pair.

Point-in-Time Recovery

Dynamic Server allows you to restore data from backup media to a specified point in time following the time of the backup. This feature enables you to restore a corrupted database to a point in time at which you know that the data was reliable. If you use ON-Bar for your backup and restore tool, refer to your *Backup and Restore Guide* for more information about this feature. If you use ON-Archive, refer to your *Archive and Backup Guide*.

Dynamic System Administration

Distributed databases require dynamic system administration tools to monitor and tune the database server. The database server provides many administration tools to help you perform these tasks easily. These tools includes the system monitoring interface (SMI), the **onperf** performance utility, command-line utilities, and the Informix Enterprise Command Center.

IECC is a graphical interface, which runs on Windows NT, for administering Dynamic Server on both UNIX and Windows NT platforms. IECC enables you to create and modify databases, tables, synonyms, views, stored procedures, and triggers. It also enables you to perform system management, event management, and data management tasks, including backups and restores. For more information about IECC, see the *Informix Enterprise Command Center User Guide*.

Distributed Data Queries

Dynamic Server allows users to query (and update) more than one database across multiple database servers within a single transaction. The database servers can reside on a single host computer or on the same network. For a list of the types of networks that the database server supports, refer to Chapter 4, "Client/Server Communications."

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 INFORMIX-Enterprise Gateway products to manipulate data in non-Informix databases. 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. These protocols are described in detail in Chapter 30, "Multiphase Commit Protocols."

UNIX

You can also use Dynamic Server in a heterogeneous environment that conforms to X/Open. For more information about using Dynamic Server within an X/Open environment, refer to the *TP/XA Programmer's Manual*. ♦

Database Server Security

Dynamic Server enforces access privileges to databases and tables through the use of the SQL statements GRANT and REVOKE. Database and table privileges are explained in the *Informix Guide to SQL: Tutorial* and the *Informix Guide to SQL: Syntax*.

In addition to this type of security, Dynamic Server offers the ability to audit database events on a database server-wide basis. Auditing, described in your *Trusted Facility Manual*, enables you to track which users performed which actions to which objects at what time. You can use this information to monitor database activity for suspicious use, deter unscrupulous users, or even act as evidence of database server abuse.

Features Beyond the Scope of Dynamic Server

As a Dynamic Server administrator, you need to know the boundaries of Dynamic Server capabilities. This section describes the tasks that lie outside the scope of the database server but are provided by your host computer, operating system, or some other product.

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 an abort. Otherwise, the database server can continue to operate, but applications cannot access the down chunk until its dbspace is restored.

No Scanning or Compression of TEXT and BYTE Data Types

Dynamic Server receives TEXT and BYTE data into an existing table in the following ways:

- From the DB-Access LOAD statement
- From the **dbload** utility
- From INFORMIX-ESQL/C programs

For information on how to write TEXT and BYTE data to a database from an application program, refer to the *INFORMIX-ESQL/C Programmer's Manual*.

The database server does not contain any mechanisms for scanning TEXT and BYTE data and inserting the data into a file, or for compression, after the data has been scanned.

Chapter

2

Overview of Database Server Administration

Who Is the Database Server Administrator? 2-3

Initial Tasks 2-4

Routine Tasks. 2-5
- Changing Database Server Operating Modes 2-5
- Backing Up Data and Logical-Log Files 2-6
- Monitoring Activity 2-6
- Checking for Consistency 2-6

Configuration Tasks 2-7
- Managing Disk Space 2-7
- Managing Database-Logging Status. 2-7
- Logical-Log Administration 2-7
- Physical-Log Administration 2-8
- Using Auditing. 2-8
- Using Mirroring 2-8
- Using Data Replication 2-9
- Managing Shared Memory. 2-9
- Managing Virtual Processors 2-10
- Managing Parallel Database Query 2-10

Administration Tasks Summary 2-11

As a database server administrator, you need to be aware of the tasks and responsibilities that fall into your domain. This chapter describes the three types of tasks you need to perform as an administrator of an Informix database server:

- Initial installation and configuration tasks
- Routine tasks that you perform on a regular basis
- Configuration tasks that you perform less frequently

If you are familiar with database server administration tasks and only need a quick review, see "Administration Tasks Summary" on page 2-11.

Who Is the Database Server Administrator?

Most database server administrative tasks require you to have the privileges accorded to the database server administrator. On UNIX systems, you must log in as user **informix** to acquire the privileges of the database server administrator. On Windows NT platforms, you must belong to the **Informix-Admin** group to administer the database server.

Important: *This manual uses the term user* ***informix*** *to refer to the database server administrator for both UNIX and Windows NT platforms. For a Windows NT platform, user* ***informix*** *means that you must be a member of the* ***Informix-Admin*** *group. Similarly, when this manual refers to user* ***root****, it means that you must be the user* ***administrator*** *on a Windows NT platform.*

Initial Tasks

When you first acquire your database server, you need to perform the following initial installation and configuration tasks:

- Plan for the database server
- Configure the operating system
- Install the database server
- Configure the database server
- Initialize the database server

If you are migrating from another Informix database server, refer to the *Informix Migration Guide*.

The following list explains each task in more detail.

1. Planning for the database server

   When you plan for the installation and use of a database server, you need to examine your priorities and resources. Refer to "Planning for the Database Server" on page 3-3 to review a list of topics.

2. Altering your Operating System

   Before you install the product, you might need to define the install directories and operating-system consoles. Refer to "Configuring the Operating System" on page 3-5 to determine if you need to alter your operating-system configuration prior to the installation of the database server product.

3. Installing the Product

   For information about how to install the product, see your *Installation Guide*.

   If you are installing another database server instance on the same physical machine, see Chapter 6, "Using Multiple Residency."

   After you install the database server, review the release notes for the operating-system version, patches to install, and other operating system information.

4. Configure the database server

   When you install the database server, many of the parameters that define the database server, such as database space, tape devices, logs, and so on, are already set to the Informix database server defaults. At this point in time, you need to configure the database server with your specific parameters, which includes defining connectivity. The database server uses **sqlhosts** to specify how clients connect to the database server. For more information on how to configure the database server, refer to Chapter 3, "Configuring the Database Server." For more information on how to define connectivity, see Chapter 4, "Client/Server Communications."

5. Initialize the database server

   When you initialize the database server, you initialize disk space and shared memory. For complete instructions on how to initialize the database server, refer to Chapter 8, "Initializing the Database Server."

Routine 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 off-line mode, so database server operating mode changes would not be a routine task.

Changing Database Server Operating Modes

As the database server administrator, your responsibilities include starting and shutting down the database server, as well as for placing it in administration mode when necessary. Chapter 7, "Managing Database Server Operating Modes," explains how to change database server modes.

Backing Up Data and Logical-Log Files

Frequent backing up of data and logical-log files ensures that the database server can be recovered in case of a failure. The manual for your backup and restore system provides you with advice and guidelines for scheduling and coordinating dbspace and logical-log backup activity. The following table lists the manuals available to help you backup and restore your system.

Tool	Manual
ON-Bar	*Backup and Restore Guide*
ON-Archive	*Archive and Backup Guide*
ontape utility	"ontape: Logging, Archives, and Restore" on page 35-95

Monitoring Activity

The Informix database server design lets you monitor every aspect of the database server such as shared-memory information, virtual processors, sessions and threads, logging activity, disk usage, and so on. Chapter 29, "Monitoring the Database Server," provides you with descriptions of the available information, instructions for how to obtain information, and suggestions for its use.

You can also use IECC to monitor database activity. For more information, see the *Informix Enterprise Command Center User Guide*.

Checking for Consistency

The database server design detects 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. Informix recommends that you perform occasional checks for data consistency. Chapter 27, "What Is Consistency Checking?" describes these checks.

Configuration Tasks

Configuration tasks are generally either setup tasks that involve initiating and maintaining functionality or performance adjustments that might become necessary as the usage pattern of your database server varies.

Managing Disk Space

If you plan to use more than one database server instance on the same computer, be aware of the issues explained in Chapter 14, "Managing Disk Space."

You are responsible for planning and implementing the layout of information managed by the database server on disks. The way you distribute the data can greatly impact the performance of the database server.

Chapter 13, "Where Is Data Stored?" explains the advantages and drawbacks of different disk configurations. Chapter 14, "Managing Disk Space," describes the actual disk-management tasks.

Managing Database-Logging Status

As a database server administrator, you can control whether a database uses transaction logging or not, and if the logging is buffered or unbuffered. You can also specify that a database is to be ANSI compliant.

Information about what these different logging options mean is in Chapter 16, "What Is Logging?" Information on how to change logging options is in Chapter 17, "Managing Database-Logging Status."

Logical-Log Administration

Although backing up logical-log files is a routine task, logical-log administration (the placement and sizing of log files, specifying high-water marks) is required even when none of your databases use transaction logging. Logical-log administration is in Chapter 18, "What Is the Logical Log?"

Instructions for creating and modifying the logical-log configuration are in Chapter 19, "Managing Logical-Log Files."

Information on backing up the logical log is included in the manuals that document the backup and restore systems. See "Backing Up Data and Logical-Log Files" on page 2-6 for a list of the manuals.

Physical-Log Administration

You can change the size and location of the physical log as part of effective disk management. For more information about the physical log, refer to Chapter 21, "Managing the Physical Log."

Using Auditing

If you use database server secure auditing, you might need to adjust a number of aspects of the auditing configuration (where audit records are stored, how to handle error conditions, and so on). You also might want to change how users are audited when you suspect that they are abusing their access privileges. These tasks, and others related to auditing, are explained in your *Trusted Facility Manual*.

Using Mirroring

Mirroring is described in Chapter 23, "What Is Mirroring?" Informix recommends that you mirror at least your root dbspace. Instructions for mirroring are provided in Chapter 24, "Using Mirroring."

Using Data Replication

Informix database servers supports high-availability data replication and Informix Enterprise Replication. High-availability data replication replicates an entire database to another database server, providing a hot standby in case of a computer crash or catastrophic failure. Enterprise Replication supports asynchronous data replication and allows you to select the exact data you require for replication. For more information on high-availability data replication, see Chapter 25, "What Is High-Availability Data Replication?" and Chapter 26, "Using High-Availability Data Replication." For information on Enterprise Replication, see the *Guide to Informix Enterprise Replication*.

Managing Shared Memory

When you manage shared memory, you might perform any or all of the following tasks:

- Change the size or number of buffers (change the size of the logical-log or physical-log buffer, or change the number of buffers in the shared-memory buffer pool)
- Change the values of shared-memory parameters (change the values)
- Change forced residency (on or off, temporarily or for this session)
- Tune checkpoint intervals
- Add segments to virtual shared memory

Use of shared memory is described in Chapter 11, "Shared Memory." Chapter 12, "Managing Shared Memory," describes the procedures to manage shared memory.

Managing Virtual Processors

The number and type of virtual processors that allow your database server to perform optimally depend on your hardware and on the type of database activity that your database server supports.

Chapter 9, "What Is Informix Dynamic Scalable Architecture?" explains what virtual processors are, and Chapter 10, "Managing Virtual Processors," explains how to change the virtual-processor configuration.

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 on-line transaction processing (OLTP) queries. The resources that you need to consider are shared memory, threads, temporary table space, and scan bandwidth. For information on parallel database query (PDQ) and how fragmentation affects the performance of PDQ, refer to your *Performance Guide*.

Administration Tasks Summary

The following tables list the initial, routine, and configuration tasks you perform to administer the database server. Figure 2-1 lists the initial tasks.

Figure 2-1
Initial Tasks Summary

Initial Tasks	Information location
Plan for database server	"Planning for the Database Server" on page 3-3
Pre-installation operating system changes	"Configuring the Operating System" on page 3-5
Install the database server	Your *Installation Guide* for Windows NT Your *Installation Guide* for UNIX
Configure the database server	"Configuring the Database Server" on page 3-7
Define connectivity	Chapter 4, "Client / Server Communications"
Initialize the database server	Chapter 8, "Initializing the Database Server"

Figure 2-2 lists the routine tasks.

Figure 2-2
Routine Tasks Summary

Routine Tasks	Information location
Change database server modes	Chapter 7, "Managing Database Server Operating Modes."
Backup data and logical log files:	
■ ON-Bar	Your *Backup and Restore Guide*
■ ON-Archive	Your *Archive and Backup Guide*
■ ontape utility	"ontape: Logging, Archives, and Restore" on page 35-95
Monitor activity	Chapter 29, "Monitoring the Database Server"
Check for data consistency	Chapter 27, "What Is Consistency Checking?"

Figure 2-3 lists the configuration tasks.

Figure 2-3
Configuration Tasks Summary

Configuration Tasks	Information location
Manage disk space	Chapter 14, “Managing Disk Space”
Analyze disk configuration	Chapter 13, “Where Is Data Stored?”
Administer the logical log: ■ Size and placement of logical logs ■ Modify logical log configuration ■ Backup logical log	 Chapter 18, “What Is the Logical Log?” Chapter 19, “Managing Logical-Log Files” *Backup and Restore Guide*
Administer the physical log	Chapter 21, “Managing the Physical Log”
Administer auditing	*Trusted Facility Manual*
Manage mirroring	Chapter 23, “What Is Mirroring?” Chapter 24, “Using Mirroring”
Replicate Data	Chapter 25, “What Is High-Availability Data Replication?” Chapter 26, “Using High-Availability Data Replication” *Guide to Informix Enterprise Replication*
Manage Shared Memory	Chapter 11, “Shared Memory” Chapter 12, “Managing Shared Memory”
Manage Virtual Processors	Chapter 10, “Managing Virtual Processors”
Manage Parallel Data Query (PDQ)	*Performance Guide*

Configuration

Section II

Chapter

3

Configuring the Database Server

Planning for the Database Server 3-3
Consider Your Priorities 3-3
Consider Your Resources 3-4

Configuring the Operating System 3-5
Operating-System Administration Facilities 3-5
Configuring Operating-System Resources 3-6

Installing the Database Server 3-7

Configuring the Database Server 3-7
Set Environment Variables 3-8
Other Environment Variables 3-9
Prepare the ONCONFIG Configuration File 3-10
Preparing the ONCONFIG FIle on UNIX 3-11
Preparing the ONCONFIG FIle on Windows NT 3-11
Required and Optional Parameters 3-12
Overview of Configuration Parameters 3-13
Allocate Disk Space 3-28
Prepare the Connectivity Information 3-28
Prepare the ON-Archive Configuration File 3-29
Prepare for Global Language Support 3-29
Evaluate Operating-System Parameters 3-29
Start the Database Server and Initialize Disk Space 3-30
Starting and Initializing on UNIX 3-30
Starting and Initializing on Windows NT 3-30
Create Dbspaces and Blobspaces 3-30

Perform Administrative Tasks 3-31
Prepare Operating-System Startup and Shutdown Scripts 3-31
Make Arrangements for Tape Management 3-33
Make Sure Users Have the Correct Environment Variables 3-33
Warn System Administrator About cron Jobs 3-34

Implementing 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 implement an Informix database server can greatly affect the performance of database operations.

The purpose of this chapter is to help you quickly start the database server with a simple configuration and provide a point of orientation for a more studied configuration process. The following topics appear in this chapter:

- Planning for the database server
- Configuring the operating system
- Installing the database server
- Configuring the database server

Planning for the Database Server

When you are planning for the database server, consider both your priorities and your resources.

Consider Your Priorities

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

- What is your highest priority, transaction speed or safety of the data?
- Will the database server usually handle many short transactions or fewer long transactions?
- Will applications on other computers use this database server instance?

- What is the maximum number of users that you can expect?
- How much help or supervision will the users require? To what extent do you want to control their database server environments?
- Are you limited by resources for space, CPU, or availability of operators?
- How much does the database server instance need to do without supervision?

Consider Your Resources

Before you start the initial database server configuration, collect as much of the required information as possible. You need the following information:

- How many disk drives are available? What are their device names? Are some of the disk drives faster than others? How many disk controllers are available?

  The first time you initialize the database server, all tables, log files, and indexes are stored in a the root dbspace on one disk drive. After the database server is running, you can move different objects to different drives. For example, you should put the most frequently used tables on the fastest drives. For more information on how to manage disk space, see Chapter 14, "Managing Disk Space."

- How many tape drives are available? What are their device names? When is an operator available to change tapes?

  You need to select the number and size of the logical-log files so that they do not fill up before a tape backup can be made. The database server keeps statistics that help you adjust these parameters after it has been running for a while. Your backup strategy also needs to take into account availability of tape drives. If you use ON-Bar as your backup and restore tool, refer to your *Backup and Restore Guide* for more information. For information on creating an archive, refer to your *Archive and Backup Guide*.

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

  The machines-notes file indicates which parameters are applicable for each platform.

- What are the host names and IP addresses of other computers on your network?

  Does your system run Network Information Service (NIS)?

  Review the machine-notes file to determine the correct parameters for each operating system.

Configuring the Operating System

You must perform several tasks to prepare your operating system for the database server. You might need the assistance of the system administrator to perform these tasks:

- Identify operating-system administration facilities
- Set up administration of the operating system
- Configure shared memory and other operating-system resources

Operating-System Administration Facilities

Each operating system has its own set of tools for administration of multiple computers and users. For example, these tools include a central console and possibly the **kerberos** utility or various platform support programs. Your operating-system administration guide indicates which of these tools are applicable for database server system administration on your hardware platform.

If your operating system uses a central console, you can use that console to manage and maintain the system from a central location. Some operating systems refer to the console as a central-control workstation. Other operating systems refer to it as an administrative workstation.

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

For details on how to set up your system console, refer to your operating-system administration guide.

Configuring Operating-System Resources

UNIX

Your database server product arrives with a machine-notes files that contains recommended values that you use to configure operating-system resources. Use these recommended values when you configure the operating system on each computer within your operating environment.♦

You can specify the amount of shared memory for the database server on some operating systems. The amount of available memory influences the values that you can choose for the shared-memory parameters. 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 operating-system kernel parameters in your database server, see Chapter 11, "Shared Memory."

WIN NT

In the Windows NT environment, Informix database servers require New Technology File Systems (NTFS). If you have an existing File Allocation Table (FAT) partition, you need to convert the partitions to NTFS. You can use the Windows NT **convert** utility as shown in the following example:

```
convert /fs:ntfs
```

♦

Installing the Database Server

Installation refers to the process of loading files from the product distribution media onto your computer and running the installation script to set up the product files.

WIN NT

In the Windows NT environment, Informix places the system files in subdirectories under the root directory. If your computer has only Windows NT installed, the root directory is **WINNT**. If your computer has Windows NT and Windows 95 installed, the root is directory is **WINDOWS**. ♦

For information about how to install the database server, see your *Installation Guide*.

If you run more than one instance of the database server on the same computer, it is called *multiple residency*. To prepare to use multiple instances of the database server, first install and configure *one* database server. Then refer to Chapter 6, "Using Multiple Residency."

Warning: *You prepare multiple residency by initializing multiple instances of the database server. Do not try to install the same version of database server more than once from the install media.*

Configuring the Database Server

After the database server is installed, it must be configured before it can be brought on-line. Most of your configuration parameters are set to the default setting during the installation of the product. However, you will want to change the configuration parameters to customize the database server for your data-processing environment.

WIN NT

In the Windows NT environment, 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. If you select **Command Line utilities** from the Informix Enterprise Command Center, the file executes automatically. You can also execute **setenv.cmd** from the command line. ♦

To configure the database server

1. Set environment variables (page 3-8).
2. Prepare the ONCONFIG configuration file (page 3-10).
3. Allocate disk space (page 3-28).
4. Prepare the connectivity configuration file (page 3-28).
5. If you will be using ON-Archive for backups, prepare the ON-Archive configuration file (page 3-29).

GLS

6. Prepare for Global Language Support (page 3-29). ♦
7. Evaluate operating-system parameters (page 3-29).
8. Start the database server and initialize disk space (page 3-30).
9. Create dbspaces and blobspaces, if desired (page 3-30).
10. Perform administrative tasks (page 3-31).

Set Environment Variables

All users must set the following environment variables before they access the database server. For more information about the Informix environment variables, see the *Informix Guide to SQL: Reference*.

- **INFORMIXDIR**

  The **INFORMIXDIR** environment variable specifies the directory where the product files are installed. Set **INFORMIXDIR** to the directory that you created for your Informix database server.

- **PATH**

  The **PATH** environment variable tells the shell the directories to search for executable files. **PATH** must include the directory where local copies of database server utilities and are stored. The following table lists this directory for each platform.

Platform	Directory
UNIX	**$INFORMIXDIR/bin**
Windows NT	**%INFORMIXDIR%\bin**

- **INFORMIXSERVER**

  The **INFORMIXSERVER** environment variable specifies the name of the default database server.

- **ONCONFIG**

  The **ONCONFIG** environment variable specifies the name of the active ONCONFIG configuration file. After you prepare the ONCONFIG configuration file, set the **ONCONFIG** environment variable to the name of the file. If the **ONCONFIG** environment variable is not present, the database server uses configuration values from one of the files that are listed in the following table.

Platform	ONCONFIG File
UNIX	**$INFORMIXDIR/etc/onconfig**
Windows NT	**%INFORMIXDIR%\etc\onconfig.std**

Tip: *Informix recommends that you set these environment variables in the appropriate start-up file for your shell or graphical-user interface.*

Other Environment Variables

All users who use database server utilities such as **onstat** must set the **ONCONFIG** environment variable to the name of the ONCONFIG configuration file. For additional environment variables, see the *Informix Guide to SQL: Reference*.

Users might need additional environment variables, such as **LC_COLLATE**, to describe their environment fully.

To set additional environment variables for each database server user, the database administrator can prepare an environment-configuration file, named **$INFORMIXDIR/etc/informix.rc** on UNIX or **%INFORMIXDIR%\etc\informix.rc** on Windows NT.

UNIX

On UNIX, to override environment variables that have been automatically set for users, each user can use a private environment-variable file, **~/.informix**, or assign new values to environment variables individually. ♦

Certain environment variables are not required for initialization, but you must set them before you can connect a client application to the database server.

UNIX

On UNIX, the following variables are required to connect applications:

- **TERMINFO**
- **INFORMIXTERM**

The **TERMINFO** and **INFORMIXTERM** environment variables specify the type of terminal interface. You might need assistance from the UNIX system administrator to set these system-dependent variables. ♦

For information about the environment variable files, see the *Informix Guide to SQL: Reference*.

Prepare the ONCONFIG Configuration File

The **onconfig.std** file contains initial setting for the configuration parameters and serves as a template for all other configuration files that you create. The following table lists the directories into which **onconfig.std** is loaded during installation.

Platform	ONCONFIG File
UNIX	**$INFORMIXDIR/etc/onconfig**
Windows NT	**%INFORMIXDIR%\etc\onconfig.std**

A sample **onconfig.std** file appears in Appendix A.

You can create and modify the ONCONFIG configuration file using a standard text editor or the ON-Monitor utility. Directions for using ON-Monitor appear in Chapter 32, "ON-Monitor for UNIX."

Preparing the ONCONFIG FIle on UNIX

To prepare the ONCONFIG configuration file with a text editor, perform the following steps:

1. Make a copy of the standard ONCONFIG file. This file is named **$INFORMIXDIR/etc/onconfig.std**. Store the new file in the same directory as the standard file, but do not modify the standard file itself. Instead, set your **ONCONFIG** environment variable to the name of your new ONCONFIG file.
2. Edit your new ONCONFIG file to modify the configuration parameters that you decide to change.

WIN NT

Preparing the ONCONFIG FIle on Windows NT

You can prepare the ONCONFIG configuration file with the **regedt32** tool or with a text editor.

To use the regedt32 tool

1. Start the **regedt32** tool.
2. Edit the **ONCONFIG: REG_SZ :ONCONFIG.1** file in the following folder:

   ```
   HKEY_LOCAL_MACHINES\SOFTWARE\Informix\OnLine
   \CurrentVersion\Environment
   ```

3. Edit variables.

   For existing variables, double-click the variable value and change the value in the String Editor window.

   For new variables, follow these steps:

   a. Choose **Edit→ Add Value**.

   b. Select `REG_SZ` as the data type.

   c. Add the variable value in the String Editor window.

To use a text editor

1. Copy the file **etc\onconfig.std** to **etc\%ONCONFIG%**.
2. Edit the new file and change the configuration parameters.

Required and Optional Parameters

Some configuration parameters must be reviewed before you initialize the database server. For an overview of all configuration parameters, see "Overview of Configuration Parameters" on page 3-13.

The following parameters *must* be reviewed and changed if necessary:

- ROOTPATH page 33-77
- SERVERNUM page 33-79
- DBSERVERNAME page 33-18

If you are using multiple communication protocols, you must set the DBSERVERALIASES configuration parameter. (See page 33-17.)

Check the following parameters if you are using the **ontape** archiving tool, or the **onunload** and **onload** utilities:

- TAPEDEV page 33-88
- LTAPEDEV page 33-48

Tip: *If you are using ON-Archive or ON-Bar, LTAPEDEV cannot be set to* ***/dev/null****.*

For information about **ontape** and ON-Archive, refer to your *Archive and Backup Guide*. For information about ON-Bar, refer to your *Backup and Restore Guide*. For information about **onunload** and **onload**, see the *Informix Migration Guide*.

Verify that the following parameters have valid pathnames:

- MSGPATH page 33-55
- CONSOLE page 33-15

After your database server is configured and running, review the configuration parameters. "Monitoring Configuration Information" on page 29-14 discusses different ways that you can examine your configuration.

Overview of Configuration Parameters

This section discusses the parameters of the ONCONFIG configuration file, grouped by function. It gives short descriptions of the parameters and refers you to more detailed discussions.

Root Dbspace

The first piece of storage that you allocate is called the root database space, or root dbspace. It stores all the basic information that describes your database server. The parameters that describe the root dbspace are as follows.

Configuration Parameter	Page	Description
ROOTNAME	33-76	You can choose any descriptive name for the ROOTNAME, but it is usually called **rootdbs**, which is its initial setting.
ROOTPATH	33-77	The ROOTPATH is the pathname of the storage allocated to the root dbspace. For information on how to choose and allocate the storage, see Chapter 14, "Managing Disk Space."
ROOTSIZE	33-78	ROOTSIZE is the amount of space allocated to the root dbspace. For information on how to choose an appropriate size for the root dbspace, see "Calculate the Size of the Root Dbspace" on page 13-30.
ROOTOFFSET	33-76	For information about when you need to set ROOTOFFSET, refer to "Allocating Raw Disk Space on UNIX" on page 14-8.

Disk Space Management Parameters

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

Configuration Parameter	Page	Description
DBSPACETEMP	33-19	DBSPACETEMP specifies a list of dbspaces that the database server uses to spread out the storage of temporary tables and the associated I/O across multiple disks.
FILLFACTOR	33-36	FILLFACTOR specifies how full index-pages to fill index pages when indexes are created.
ONDBSPACEDOWN	33-66	ONDBSPACEDOWN defines how the database server treats a disabled dbspace that is not a critical dbspace.

Database Server Identification Parameters

The identification parameters provide the unique identification of the database server. The parameters are as follows.

Configuration Parameter	Page	Description
DBSERVERNAME	33-18	DBSERVERNAME specifies the name of the database server. You use the dbservername in **sqlhosts** and with the **INFORMIXSERVER** environment variable. For a complete description of DBSERVERNAME, see "The DBSERVERNAME Configuration Parameter" on page 4-51.
DBSERVERALIASES	33-17	DBSERVERALIASES specifies a list of alternative dbservernames used to denote multiple communication protocols. For information about DBSERVERALIASES, refer to "The DBSERVERALIASES Configuration Parameter" on page 4-51
SERVERNUM	33-79	SERVERNUM specifies a unique integer for each database server instance. It must be unique for each database server on your local computer but does not need to be unique across your network. For more information about SERVERNUM, refer to "The Role of the SERVERNUM Configuration Parameter" on page 5-5.

Mirroring Parameters

Mirroring allows very fast recovery from a disk crash while the database server remains in on-line 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. The following parameters describe mirroring of the root dbspace.

Configuration Parameter	Page	Description
MIRROR	33-53	MIRROR defines whether mirroring is enabled or disabled.
MIRRORPATH	33-54	MIRRORPATH specifies the full pathname of the chunk that mirrors the initial chunk of the root dbspace.
MIRROROFFSET	33-54	MIRROROFFSET specifies the offset into the device that servers as the mirror for the initial root dbspace chunk.

For more information on the performance implications of mirroring, see Chapter 23, "What Is Mirroring?"

Logical Logging Parameters

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 describe logical logging.

Configuration Parameter	Page	Description
LOGBUFF	33-41	The LOGBUFF parameter determines the amount of shared memory reserved for the buffers that hold the logical-log records until they are flushed to disk. You can use the initial setting for LOGBUFF unless your database server has an unusually large number of transactions. For information on how to tune the logical-log buffer, see "Logical-Log Buffer" on page 11-22.
LOGFILES	33-42	The logical-log records are stored on disk in logical-log files until they are backed up to tape. LOGFILES specifies the number of logical-log files.
LOGSIZE	33-43	LOGSIZE is the size of each logical-log file.
LOGSMAX	33-44	LOGSMAX is the maximum (not the actual) number of log files that you expect to have. The number and size of the logical-log files needed depends on the activity of your database server and the frequency of log-file backups. For more information about these parameters, refer to "What Should Be the Size and Number of Logical-Log Files?" on page 18-11.
LTXHWM	33-51	The long-transaction high-water mark parameter specifies the percentage of the available logical log that can be used before the database server takes moderate action to avoid the undesirable effects of reaching the point of LTXEHWM, the long-transaction exclusive-access high-water mark.
LTXEHWM	33-50	LTXEHWM is the point at which the database server takes drastic action. For more information on high-watermark parameters, see "Avoiding Long Transactions" on page 18-18.

Physical Logging Parameters

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

Configuration Parameter	Page	Description
PHYSFILE	33-71	PHYSFILE specifies the size of the physical log.
PHYSDBS	33-70	PHYSDBS specifies the name of the dbspace where the physical log resides. When disk space is first initialized, the physical log must reside in the root dbspace. For more information, see Chapter 20, "What Is Physical Logging?"
PHYSBUFF	33-69	The PHYSBUFF parameter determines the amount of shared memory reserved for the buffers that serve as temporary storage space for pages about to be modified.

Archiving and Logical-Log Backup Parameters

To create dbspace backups and make logical log backups for data managed by the database server, you can use ON-Bar or ON-Archive, as well as the ontape utility. For information on ON-Bar, see your *Backup and Restore Guide*. For information on ON-Archive, see your *Archive and Backup Guide*. If you use the **ontape** utility, use the following parameters to archive data and logical log backups.

Configuration Parameter	Page	Description
TAPEDEV	33-88	TAPEDEV specifies tape devices.
TAPEBLK	33-87	TAPEBLLK specifies the block size of the tape device.
TAPESIZE	33-90	TAPESIZE specifies the maximum amount of data that should be written to each tape.

(1 of 2)

Configuration Parameter	Page	Description
LTAPEDEV	33-48	LTAPEDEV specifies tape devices.
LTAPEBLK	33-47	LTAPEBLK specifies the block size of the tape device.
LTAPESIZE	33-49	LTAPESIZE specifies the maximum amount of data that should be written to each tape.

(2 of 2)

Message-Log Parameters

The database server message logs provide information about how the database server is functioning. The following parameters give the pathnames of the message logs.

Configuration Parameter	Page	Description
CONSOLE	33-15	CONSOLE specifies the pathname destination for console messages.
MSGPATH	33-55	MSGPATH is the pathname of the database server message log. The database server writes status messages and diagnostic messages to the log. Monitor this file regularly.

Shared-Memory Parameters

The following shared-memory parameters are important to database server performance. Use these parameters to control how the database server allocates space in shared memory.

Configuration Parameter	Page	Description
BUFFERS	33-12	BUFFERS is the number of shared-memory buffers available to the database server. For more information about shared-memory buffers, refer to "Shared-Memory Buffer Pool" on page 11-19.
LBU_PRESERVE	33-38	LBU_PRESERVE specifies the logs-full high-water mark. This feature preserves log space for administrative tasks. For more information about the logs-full high-water mark, refer to "Preserving Log Space for ON-Archive Tasks" on page 18-8.
LOCKS	33-39	LOCKS specifies the maximum number of locks available to user processes during transaction processing. You can use the initial setting for LOCKS.
RESIDENT	33-74	Some systems allow you to specify that the resident portion of shared memory must stay (be resident) in memory at all times. The RESIDENT parameter specifies whether shared-memory residency is enforced. If forced residency is not an option on your computer, this parameter is ignored. For more information about residency, see "Setting Operating-System Shared-Memory Configuration Parameters" on page 12-3.
STACKSIZE	33-85	STACKSIZE specifies the stack size for user threads. The use of stacks is discussed in "Stacks" on page 11-32.
CKPTINTVL	33-13	CKPTINTVL is the maximum time allowed to elapse before a checkpoint. Use the initial settings for the initial configuration.

Shared-Memory Buffer Pool Parameters

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

Configuration Parameter	Page	Description
LRUS	33-45	The LRUS (Least Recently Used) queues manage the shared-memory pool of pages (memory spaces) used by the database server.
LRU_MAX_DIRTY	33-46	LRU_MAX_DIRTY specifies the percentage of modified pages in the LRU queues that, when reached, flags the queue to be cleaned.
LRU_MIN_DIRTY	33-46	LRU_MIN_DIRTY specifies the percentage of modified pages in the LRU queues that, when reached, flags the page cleaners that cleaning is no longer mandatory.
CLEANERS	33-14	CLEANERS controls the number of threads used to flush pages to disk and return the pages to the shared-memory pool.
RA_PAGES	33-72	RA_PAGES specifies the number of disk pages to attempt to read ahead during sequential scans of data or index records.
RA_THRESHOLD	33-73	RA_THRESHOLD is used with RA_PAGES when the database server reads during sequential scans of data and index records.

Shared-Memory Size-Allocation Parameters

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

Configuration Parameter	Page	Description
SHMADD	33-80	SHMADD specifies the increment of memory that is added (if possible) when the database server requests more memory.
SHMBASE	33-81	SHMBASE is the shared-memory base address and is computer dependent. Do not change its value.
SHMTOTAL	33-82	SHMTOTAL specifies the maximum amount of memory that the database server is allowed to use.
SHMVIRTSIZE	33-83	SHMVIRTSIZE specifies the size of the first piece of memory attached by the database server.

Fragmentation and Parallel Data Query Parameters

Use the following parameters to control which queries are processed as decision-support queries and also to control the amount of resources that the database server allocates to decision-support queries.

Configuration Parameter	Page	Description
DATASKIP	33-16	Use DATASKIP to control whether the database server skips an unavailable table fragment.
DS_MAX_QUERIES	33-26	DS_MAX_QUERIES is the maximum number of queries that can run concurrently.
DS_MAX_SCANS	33-27	DS_MAX_SCANS limits the number of PDQ scan threads that the database server can execute concurrently.
DS_TOTAL_MEMORY	33-28	DS_TOTAL_MEMORY specifies the amount of memory available for PDQ queries.
MAX_PDQPRIORTY	33-52	MAX_PDQPRIORITY imposes limits on the amount of resources that a query can use.
OPTCOMPIND	33-68	OPTCOMPIND advises the optimizer on an appropriate join strategy for your applications.

For detailed information on fragmentation and PDQ, see your *Performance Guide*.

Virtual Processor Parameters

Use the following parameters to control how database server processes work on a multiprocessor computer.

Configuration Parameter	Page	Description
AFF_NPROCS	33-9	AFF_NPROCS specifies the number of CPUS to which the database server can bind CPU virtual processors on computers that support processor affinity.
AFF_SPROC	33-10	AFF_SPROC specifies the CPU, starting with 0, on which the database server starts binding CPU virtual processors to CPUs.
MULTIPROCESSOR	33-56	MULTIPROCESSOR specifies the appropriate type of locking.
NETTYPE	33-57	NETTYPE provides tuning options for each communications protocol.
NOAGE	33-61	NOAGE specifies whether priority aging should be in effect
NUMAIOVPS	33-62	NUMAIOVPS specifies the number of virtual processors for AIO class threads.
NUMCPUVPS	33-63	NUMCPUVPS specifies the number of virtual processors for CPU class threads.
SINGLE_CPU_VP	33-84	SINGLE_CPU_VP can specify that the database server is using only one processor and allow the database server to optimize for that situation.

Time Interval Parameters

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

Configuration Parameter	Page	Description
DEADLOCK_TIMEOUT	33-21	DEADLOCK_TIMEOUT specifies the amount of time that the database server waits for a shared-memory resource during a distributed transaction.
TXTIMEOUT	33-91	TXTIMEOUT is the amount of time a participant the database server waits to receive a commit instruction during a two-phase commit. TXTIMEOUT and DEADLOCK_TIMEOUT apply only to transactions that are taking place over a network.
USEOSTIME	33-92	USEOSTIME controls the granularity of the time reported by the database server.

Restore Parameters

Use the following parameters to control the number of threads that the database server allocates to off-line and on-line logical memory.

Configuration Parameter	Page	Description
OFF_RECVRY_THREADS	33-64	OFF_RECVRY_THREADS is the number of recovery threads used in logical recovery when the database server is off-line (during a cold restore). This number of threads is also used to roll forward logical-log records in fast recovery.
ON_RECVRY_THREADS	33-65	ON_RECVRY_THREADS is the maximum number of recovery threads that the database server uses for logical recovery when the database server is on-line (that is, during a warm restore).

High-Availability Data Replication Parameters

The high-availability data replication parameters control the behavior of a pair of high-availability data replication servers. The following parameters refer to data replication.

Configuration Parameter	Page	Description
DRAUTO	33-22	DRAUTO determines how a secondary database server reacts to a high-availability data-replication failure.
DRINTERVAL	33-23	DRINTERVAL specifies the maximum time interval in seconds between flushing of the data-replication buffer.
DRLOSTFOUND	33-24	DRLOSTFOUND specifies the pathname 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	33-25	Use DRTIMEOUT to specify 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.

Error Dump Parameters

The following parameters control the types and location of core dumps that are made if the database server fails.

Configuration Parameter	Page	Description
DUMPCNT	33-32	DUMPCNT specifies the number of assertion failures for which a single thread dumps shared memory or generates a **core** file by calling the **gcore** utility.
DUMPCORE	33-33	DUMPCORE controls whether assertion failures cause a virtual processor to dump core.
DUMPDIR	33-34	DUMPDIR specifies a directory where dumps of shared memory, **gcore** files, or messages from a failed assertion are placed.
DUMPGCORE	33-34	DUMPGCORE is used with operating systems that support the **gcore** utility.
DUMPSHMEM	33-35	DUMPSHMEM indicates that shared memory should be dumped on an assertion failure.

Event Alarm Parameter

The database server can execute a file if a noteworthy event occurs. The location of the executable file is specified by the ALARMPROGRAM parameter.

Configuration Parameter	Page	Description
ALARMPROGRAM	33-11	Set ALARMPROGRAM to the full pathname of an executable file that you write and that the database server executes when noteworthy events occur.

Optical Subsystem Parameters

The STAGEBLOB parameter is used only by the Optical Subsystem.

Configuration Parameter	Page	Description
OPCACHEMAX	33-67	OPCACHEMAX specifies the size of the memory cache for the Optical Subsystem.
STAGEBLOB	33-86	Use this parameter only if you are storing TEXT or BYTE DATA TYPES on optical storage with Optical Subsystem.

Allocate Disk Space

Before you allocate the disk space, study the information about disk space in Chapter 13, "Where Is Data Stored?" For information about allocating raw disk space, refer to "Allocating Raw Disk Space on UNIX" on page 14-8. If you want to use cooked disk space, you can follow the instructions from "Allocating a File for Disk Space on UNIX" on page 14-7.

Prepare the Connectivity Information

On UNIX platforms, the **sqlhosts** file contains connectivity information that allows an Informix client application to connect to any Informix database server on the network. On Windows NT platforms, the **sqlhosts** entry in the HKEY_LOCAL_MACHINE registry contains the same connectivity information. In both cases, the **sqlhosts** data for a particular database server includes the database server name, the type of connection a client can use to connect to it, the name of the host computer or node on which it runs, and the service name by which is known.

You must prepare the **sqlhosts** file or registry even if both the client application and the database server are on the same computer or node.

Before you continue to step 5, define your connectivity with the instructions in Chapter 4, "Client/Server Communications."

Prepare the ON-Archive Configuration File

If you use ON-Archive to back up and restore database server data, you must set parameters in the ON-Archive configuration file, **config.arc**. It describes, among other things, the devices used for different archiving and backup tasks. Your *Archive and Backup Guide* describes the parameters for the **config.arc** file. Appendix A of this book describes additional ON-Archive configuration files.

GLS

Prepare for Global Language Support

GLS lets you create databases using the diacritics, collating sequences, and monetary and time conventions of the language you select. GLS does not have any ONCONFIG configuration parameters, but you must set the appropriate environment variables. For more information about GLS, see the *Informix Guide to GLS Functionality*.

Evaluate Operating-System Parameters

UNIX

Your database server product arrives with Machine Notes that contain recommended values for operating system configuration parameters. Compare the values in this file with your current operating-system configurations. ♦

The amount of memory available influences the values you can choose for the shared-memory parameters. In general, increasing the space available for shared memory enhances performance. You might also need to increase the number of locks and semaphores.

If the recommended values for the database server differ significantly from your current environment, consider modifying your operating-system settings.

For additional information that describes the role of the operating-system kernel parameters in the database server, see "Setting Operating-System Shared-Memory Configuration Parameters" on page 12-3.

Start the Database Server and Initialize Disk Space

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

UNIX

Starting and Initializing on UNIX

To bring the database server to on-line mode, type the following command at the system prompt:

```
oninit
```

If you are starting a *new* instance of

a database server, use the **oninit** command with an **-i** flag to initialize the disk space and bring the database server into on-line mode.

You can also initialize disk space using ON-Monitor. For more information about ON-Monitor, refer to Chapter 32, "ON-Monitor for UNIX."

For a description of the types of initialization and associated commands, see "Managing Database Server Operating Modes" on page 7-1.

WIN NT

Starting and Initializing on Windows NT

In Windows NT, the database server runs as a service. Use the **Service** control application to bring the database server to on-line mode. For a description of the types of initialization and the associated commands, see "Managing Database Server Operating Modes" on page 7-1.

Create Dbspaces and Blobspaces

Now that the database server is initialized, you can create dbspaces and blobspaces as desired. For a description of dbspaces and blobspaces, see Chapter 13, "Where Is Data Stored?" For information on how to allocate and manage dbspaces and blobspaces, see Chapter 14, "Managing Disk Space."

Perform Administrative Tasks

After you initialize the database server, you need to perform the following administrative tasks:

- Prepare the operating-system registry or scripts to automatically start and stop the database server.
- Make arrangements for tape management.
- Make sure users have the correct environment variables.
- Warn the operating-system administrator about **cron** jobs.

Prepare Operating-System Startup and Shutdown Scripts

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

WIN NT

Preparing for Database Server Automatic Startup on Windows NT

Windows NT does not support multiple users on one computer.

To start the database server automatically when Windows NT starts, perform the following actions from the **Service** control application window:

1. In the **Status Type** dialog box, select **Automatic**.
2. In the **Log On As** dialog box, select **This Account** and verify that `informix` is in the text box.

Inform the **Service** control application of a new Informix password.

UNIX

Preparing the UNIX Startup and Shutdown Scripts

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.

To prepare the UNIX startup script

1. Set the **INFORMIXDIR** environment variable to the full pathname 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 on-line mode.
6. 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.
7. If you are using ON-Archive, you might want to start **oncatlgr**. For more information about ON-Archive, refer to your *Archive and Backup Guide*.

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

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 script

1. Set the **INFORMIXDIR** environment variable to the full pathname 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 Immediate-Shutdown and takes the database server off-line.

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

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.

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

Make Arrangements for Tape Management

When you plan your data dbspace and logical-log backup schedule, as discussed in your *Archive and Backup Guide*, take into account the availability of tape devices to manage the data and the availability of operators to perform backups. If you use ON-Archive, you can take advantage of its ability to perform unattended operations.

Make Sure Users Have the Correct Environment Variables

Make sure that every user of an Informix product has the correct environment variables. Environment variables are discussed in the *Informix Guide to SQL: Reference*.

Each user must set the following environment variables before accessing the database server:

- **INFORMIXSERVER**
- **INFORMIXDIR**
- **PATH**

In addition, all users who use utilities such as **onstat** must set the **ONCONFIG** environment variable to the name of the ONCONFIG configuration file.

Warn System Administrator About cron Jobs

UNIX

The database server creates the **.inf.*servicename*** and / or **VP.*servername.xxC*** files in the **/INFORMIXTMP** directory. Some operating systems run **cron** jobs that routinely delete all files from the **/INFORMIXTMP** directory. For information about these files, see Appendix A. ♦

WIN NT

To run cron jobs under Windows NT, use the **at** command from the command line. ♦

Chapter

4

Client/Server Communications

What Is Client/Server Architecture? 4-3
What Is a Network Protocol? 4-4
What Is a Network Programming Interface? 4-5
Windows NT Network Domains 4-5

What Is a Connection? 4-6
What Is a Multiplexed Connection? 4-7

What Connections Does the Database Server Support? 4-8
Network Connections 4-8
Local Connections 4-9
Shared-Memory Connections 4-10
Stream-Pipe Connections 4-11
Named-Pipe Connections 4-11
Local Loopback Network Connections 4-12
What Interface/Protocol Combinations Are Available on Your Platform? 4-13

What Are Communications Support Services? 4-13
What Is a Communications Support Module? 4-14
How to Configure the DCE-GSS Communications Support Module on UNIX 4-15
Installing and Configuring DCE Components for the DCE-GSS CSM 4-15
Describing and Specifying the CSM 4-16

Connectivity Files 4-17
Network-Configuration Files 4-17
TCP/IP Connectivity Files on UNIX 4-17
TCP/IP Connectivity Files on Windows NT 4-19
IPX/SPX Connectivity Files 4-22

Network-Security Files 4-22
The hosts.equiv File 4-22
The .netrc File 4-23
The $INFORMIXDIR/etc/concsm.cfg file 4-25
The sqlhosts File or Registry 4-27
The sqlhosts File 4-27
The sqlhosts Registry 4-28
sqlhosts File and Registry Fields. 4-30
The dbservername Field and Key Name Field 4-31
The nettype and Protocol Field 4-31
The hostname and host Field 4-34
The servicename and Service Field 4-40
The options Field 4-43

ONCONFIG Parameters for Connectivity 4-50
The DBSERVERNAME Configuration Parameter 4-51
The DBSERVERALIASES Configuration Parameter 4-51

Environment Variables for Network Connections 4-52

Examples of Client/Server Configurations 4-53
Using a Shared-Memory Connection 4-54
Using a Local Loopback Connection. 4-55
Using a Network Connection 4-56
Using Multiple Connection Types 4-57
Accessing Multiple Database Servers 4-59
Using the Relay Module 4-60
A Relay Module Configuration with Three Database Servers 4-62
Using a Post-6.0 Client Application with a 5.x Database Server 4-63

This chapter explains the concepts and terms that you need to configure client / server communications. The chapter is divided into the following parts:

- Description of client / server architecture
- Database server connection types
- Communications services
- Connectivity files
  - Network-configuration files
  - Network-security files
- **sqlhosts** information
- Description of ONCONFIG connectivity parameters
- Examples of client / server configurations

What Is Client/Server Architecture?

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, although they can also reside on the same computer. Client applications issue requests for services and data from the server. The 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. These terms are defined in detail in the following sections.

What Is a 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 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 according to the rules of a protocol when it is sent from client to server and from 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 NT is WINSOC (sockets programming interface).

The power of a network protocol lies in its ability to enable client/server communication even though the client and 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 "What Interface/Protocol Combinations Are Available on Your Platform?" on page 4-13.

To specify which protocol the database server uses, set the PROTOCOL field in the **sqlhosts** file or registry. For more information, see "The sqlhosts File or Registry" on page 4-27.

What Is a 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 server are the applications that call the routines in the sockets or TLI application-programming interface. Clients and database servers both make use of 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 server, or vice versa.

WIN NT

Windows NT Network Domains

Windows NT network technology enables you to create network *domains*. A domain is a group of connected Windows NT 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, when the network administrator creates or changes a user account, he or she 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 NT server, a user on another Windows NT 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 trusting domain.

Important: *The Informix trusted client mechanism has nothing to do with the trust relationship that you can establish between Windows NT domains. Therefore, even if a client connects from a trusted Windows NT domain, Informix requires the user to have an account in the domain on which the database server is running. For more information on how the database server authenticates clients, see* *"What Are Communications Support Services?" on page 4-13* and *"Network-Security Files" on page 4-22*.

What Is a 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 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 Service 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 with a database server with either the CONNECT or DATABASE SQL statement. For example, to connect to the database server **my_server**, your application might contain this form of the CONNECT statement:

```
CONNECT TO '@myserver'
```

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

What Is a Multiplexed Connection?

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 the 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, you must include in the ONCONFIG file a special NETTYPE parameter that has a value of `SQLMUX`. To configure the automatic use of multiplexed connections by clients, you must also specify an option value of `m=1` on the **sqlhosts** parameter for the database server connection. 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 registry are configured appropriately. For information on the NETTYPE parameter, refer to "NETTYPE" on page 33-57. For information on the **sqlhosts** parameter for a database server connection, refer to "The sqlhosts File or Registry" on page 4-27.

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 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.

What Connections Does the Database Server Support?

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

Connection Type	Windows NT	UNIX	Local	Network
Sockets	X	X	X	X
TLI (transport layer interface)		X	X	X
Shared memory		X	X	
Stream pipe		X	X	
Named pipe	X		X	

You can make local connections with Sockets and TLI by creating a local loopback connection. For information on local loopback connections, refer to "Local Loopback Network Connections" on page 4-12.

The following sections describe database server connection types in more detail.

Network Connections

To establish a connection between a client on one computer and a database server on another computer, you must use a combination of a network interface and a network protocol to create a network connection.

The database server supports the following types of interface / protocol combinations for network connections.

Interface	Network Protocol	Windows NT	UNIX
Sockets	TCP / IP	X	X
TLI (transport layer interface)	TCP / IP		X
TLI (transport layer interface)	IPX / SPX		X

WIN NT

On Windows NT, Informix uses the Sockets interface to support the TCP / IP communication protocol for client-to-server and server-to-server connections. ♦

For information on how to implement a network connection, refer to "Connectivity Files" on page 4-17, "ONCONFIG Parameters for Connectivity" on page 4-50, and "Environment Variables for Network Connections" on page 4-52.

Local Connections

A *local connection* is a connection between a client and the database server on the same computer. The following table illustrates the types of local connections that a client can establish with the database server.

Local Connection Type	Windows NT	UNIX
Shared memory		X
Stream pipe		X
Named pipe	X	
Local loopback	X	X

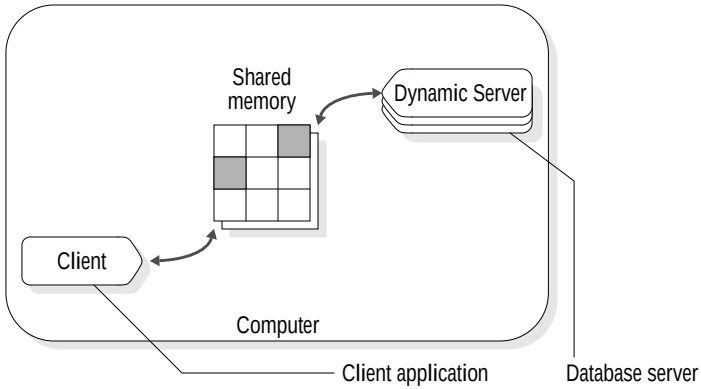
The following sections describe these types of local connections.

UNIX

Shared-Memory Connections

A *shared-memory connection* uses an area of shared-memory as the *channel* through which the client and server communicate with each other. Figure 4-1 illustrates a shared-memory connection.

Figure 4-1
A Shared-Memory Connection



Shared memory provides very fast access to a database server, but it poses some security risks. Errant or untrusted 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 application if you use network communication.

For information about the portion of shared memory that the database server uses for client / server communications, refer to "The Communications Portion of Shared Memory" on page 11-34. For additional information, you can also refer to "Where the Client Attaches to the Communications Portion" on page 11-11.

UNIX

Stream-Pipe Connections

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. You can also use stream-pipe connections for distributed database operations when both database servers are on the same computer.

Advantages of Stream-Pipe Connections

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.
- Stream-pipe connections allow distributed transactions between database servers that are on the same computer.

Disadvantages of Stream-Pipe Connections

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.

WIN NT

Named-Pipe Connections

Named pipes are application-programming interfaces (APIs) for bidirectional interprocess communication (IPC) on Windows NT. Named-pipe connections provide a high-level interface to the network 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 and writing from a file system.

Informix supports named pipes for local connections to database servers on Windows NT.

For information on how to specify a named-pipe connection, refer to "The sqlhosts File or Registry" on page 4-27. For additional information, regarding the impact of the NETTYPE configuration parameter, refer to "NETTYPE" on page 33-57.

Local Loopback Network 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.

Figure 4-2 shows how a local loopback connection appears to the client application and to the database server. Data appears to pass from the client application, out to the network, and then back in again to the database server. Data can also pass from the database server to the client application in a similar manner. That is, the data passes from the database server, out to the network, and then back in again to the client application.

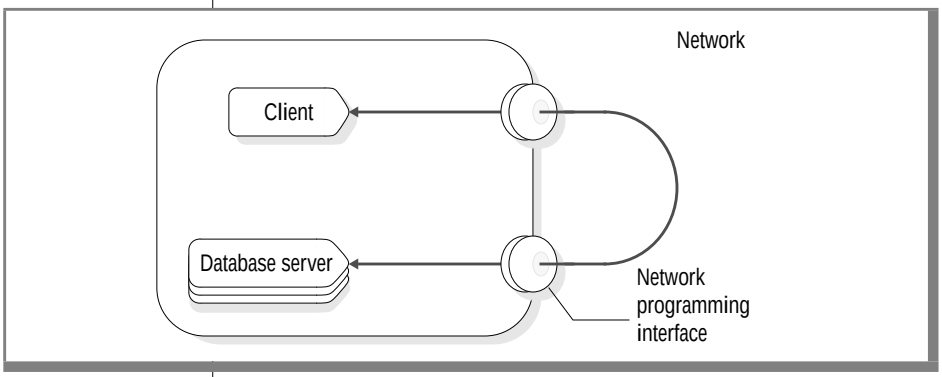


Figure 4-2
Conceptual Illustration of Local Loopback Connection

You can think of a local loopback connection as shown in Figure 4-2, but the diagram in Figure 4-3 on page 4-13 is a more accurate representation of local loopback. Figure 4-3, which illustrates a local loopback connection, differs from the shared-memory diagram in Figure 4-1 on page 4-10 only in the type of connection between the client application and the database server.

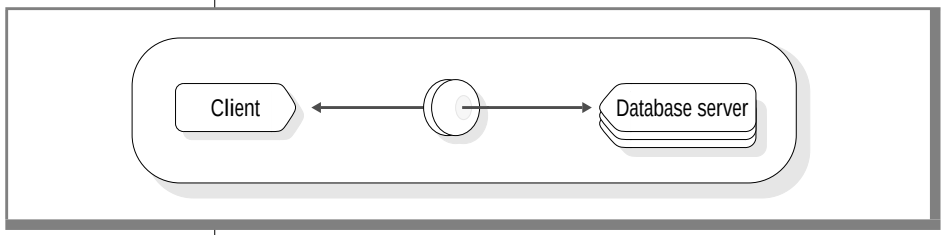


Figure 4-3
A Simple Local Loopback Connection

UNIX

What Interface/Protocol Combinations Are Available on Your Platform?

On many UNIX platforms, the database server supports multiple network programming interfaces. To check which interface / protocol combinations the database server supports for your operating system, check the machine notes. The section "Machine Specific Notes" describes interface / protocol combinations that are available on your platform, similar to the following example.

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

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

              Berkeley sockets using TCP/IP
```

What Are Communications Support Services?

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

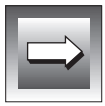
- Authentication
- Message integrity
- Message privacy

Communications support services can also include other processing such as data compression or traffic-based accounting, for example.

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. The database server provides a default method of authentication, which is described in the section "Network-Security Files" on page 4-22. You do not need to perform any procedures to use the default authentication policy. The database server uses the default authentication policy when you do not specify a communications support module.

Message integrity ensures that communication messages are intact and unaltered when they arrive at their destination.

Message privacy protects messages, usually by encryption and decryption, from viewing by unauthorized users during transmission.



Important: *Communications support services apply only to the connection between the database server and a client application. Communication support services do not apply to a connection between the database server and another Informix database server for operations such as distributed queries.*

UNIX

What Is a Communications Support Module?

On UNIX, Informix Dynamic Server lets you add external communications support services in the form of a *communications support module* (CSM). Dynamic Server includes a CSM, called DCE-GSS, which provides access to the DCE 1.1 security service for user authentication, message integrity and message privacy. For information on whether the DCE-GSS CSM is available on your platform, refer to the machine-notes file, which is described in "Documentation Notes, Release Notes, Machine Notes" on page 15.

The DCE-GSS CSM uses the DCE 1.1 security service to provide communications support services. Informix recommends that you consult with the manufacturer for your operating system about obtaining DCE 1.1. If the manufacturer for your operating system does not provide DCE 1.1, you might need to obtain it from a third-party vendor.

The DCE-GSS CSM is a shared library that the database server loads dynamically. The library is installed in the **$INFORMIXDIR/lib/csm** directory as **libixdce.XX**. The **.XX** extension is the platform-specific shared-library filename extension. If the actual extension for a shared library is **.so**, for example, the name of the file would be **libixdce.so.**

The following section describes how to configure the communications support module.

UNIX

How to Configure the DCE-GSS Communications Support Module on UNIX

To use the DCE-GSS CSM, perform the following basic steps:

1. Install and configure the DCE components according to the third-party documentation.
2. Describe the communications support module in the **concsm.cfg** file. For more information on how to describe the **concsm.cfg** file, refer to "Describing and Specifying the CSM" on page 4-16.
3. Create an entry in the options field of the **sqlhosts** file to specify that the connection will use the communications support module. For more information on specifying the communications support module in the options field of the **sqlhosts** file, refer to "The Communications Support Module Option" on page 4-45

Installing and Configuring DCE Components for the DCE-GSS CSM

Follow these basic steps to install and configure the DCE components:

1. Install and configure the DCE clients and servers properly.
2. Create a DCE principal name for each database user and for each database server name. DCE security can only authenticate DCE principals. Consequently, database users and database server names must correspond to DCE principal names. Informix recommends, for consistency, that you obtain this coordination by assigning principal names that match database user (UNIX) names and database server names. For more information on assigning principal names, refer to the DCE documentation.

3. Create a *keytab* file for database server principals, and set the **INFORMIXKEYTAB** environment variable to specify the location of the file. For more information about the **INFORMIXKEYTAB** environment variable, see the *Informix Guide to SQL: Reference*.

   The keytab file is a DCE security-related file. The keytab file contains key tables that store keys, each of which contains a principal name, type, version, and value. For more information on keytab files, refer to the third-party DCE documentation.

   The database server requires the **INFORMIXKEYTAB** environment variable and the keytab file to find the key to register the server and acquire the credential for it. A client application requires the **INFOMRMIXKEYTAB** environment variable and the keytab file if the user did not login with the current operating-system user name as the principal name prior to starting the application or did not explicitly provide a credential. In these cases, the keytab file provides a way to find the key and acquire the credential for the user.

Describing and Specifying the CSM

You must describe the CSM in the **concsm.cfg** file. The **concsm.cfg** file resides in the **$INFORMIXDIR/etc** directory by default. If you want to store the file somewhere else, you can override the default location by setting the **INFORMIXCONCSMCFG** environment variable to the full pathname of the new location. For information on setting the **INFORMIXCONCSMCFG** environment variable, refer to the *Informix Guide to SQL: Reference*.

For information on how to describe the CSM in the **concsm.cfg** file, refer to "The $INFORMIXDIR/etc/concsm.cfg file" on page 4-25. For information on specifying the CSM for a connection in the **sqlhosts** file, refer to "The options Field" on page 4-43.

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 registry

Network-Configuration Files

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

UNIX

TCP/IP Connectivity Files on UNIX

When you configure the database server to use the TCP / IP network protocol, you use information from the network-configuration files **/etc/hosts** and **/etc/services** to prepare the **sqlhosts** file. The **/etc/hosts** and **/etc/services** are UNIX files that the network administrator maintains. When you add a host, or a software service such as a database server, you need to inform the network administrator so that he or she can make sure the information in these files is accurate.

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

The /etc/hosts File

The **/etc/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
- Host name
- Host aliases (optional)

Although the length of the host name is not limited in the **/etc/hosts** file, it is limited to 255 characters in the **sqlhosts** file. Figure 4-8 on page 4-35 includes a sample **hosts** file.

The /etc/services File

The **/etc/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
- Port number and protocol
- 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 Informix client / server products. The **aliases** field is optional. For example, an **/etc/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 client-application connection requests. Figure 4-13 on page 4-41 includes a sample **services** file.

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 command line:*

```
ypcat hosts ; ypcat services.
```

For information about the **/etc/hosts** and **/etc/services** files, refer to the UNIX manual pages for hosts and services.

WIN NT

TCP/IP Connectivity Files on Windows NT

Perform the following steps to configure the database server to use the TCP/IP network protocol on Windows NT:

1. Configure the network interface (or adapter) card.
2. Install and configure the Windows NT TCP/IP network software package.
3. Create an entry in the **sqlhosts** registry.

For information on installing a network interface card for TCP/IP, consult your computer hardware documentation. To configure the network card and to install and configure the TCP/IP network software, use the Network application in the Windows NT Control Panel. For specific directions on configuring the adapter and installing the TCP/IP network software, see your Windows NT documentation.

You use information from the **hosts** and **services** network-configuration files to prepare the **sqlhosts** registry for the TCP/IP network protocol:

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

This chapter refers to these files as **\etc\hosts** and **\etc\services**, respectively.

Whenever you add a computer or a software service such as Informix Dynamic Server, you need to inform the network administrator so that he or she can make sure that the information in these files is accurate.

The \etc\hosts File

The **\etc\hosts** file needs a single entry for each computer on the network that uses an Informix client/server product. Alternately, you can configure TCP/IP to use the Domain Name Service (DNS) for host name resolutions. For information about these files, refer to your operating-system documentation. Each line in the **\etc\hosts** file contains the following information:

- Internet address (or ethernet card IP address)

  The Dynamic Host Configuration Product (DHCP) assigns IP addresses dynamically 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 should be transparent to the database server.

- Host name

  Although the length of the host name is not limited in the **\etc\hosts** file, it is limited to 255 characters in the **sqlhosts** registry.

- Host aliases (optional)

The \etc\services File

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

- Service name

  The **sqlhosts** registry uses this name to determine the port number and protocol.

- Port number and protocol

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

- Aliases (optional)

For example, an **\etc\services** file might include the following entry for an instance of the database server:

```
server2     1526\tcp
```

This entry makes **server2** known as the service name for TCP port 1526. The database server can then use this port to handle client-application connection requests.

Tip: *You can replace the host name with the corresponding IP addresses from the hosts file, and you can replace the service name with the appropriate port number from the services file. This action might result in faster connection times for client applications.*

Configuring Multiple Ports for TCP/IP

To take advantage of multiple ethernet cards, add two entries in the **sqlhosts** registry for TCP/IP connections.

To configure multiple ports for TCP/IP

1. Make an entry in the **\etc\services** file for a port of each ethernet card, as in the following example:

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

   All ports in use on a single ethernet card must be unique. Separate ethernet cards can utilize the same or different port numbers. You might want to use the same port number on each ethernet card because you are connecting to the same database server (in this scenario, the service name is the same).

2. Put one entry per ethernet card in the **\etc\hosts** file with a separate address, as in the following example:

   ```
   192.147.104.19          svc8
   192.147.104.20          svc81
   ```

3. In the **sqlhosts** registry, put one entry per ethernet card, as in the previous example.

4. In the onconfig file, enter DBSERVERNAME for one of the ethernet card **sqlhosts** entries and a DBSERVERALIASES for the other ethernet card **sqlhosts** entry.

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

UNIX

IPX/SPX Connectivity Files

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

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 platforms, this file resides in the **/etc** directory. On Windows NT platforms, this file resides in the **%WINDIR%\system32\drivers\etc** directory.

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 full domain name, as the following example shows:

```
viking.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.

UNIX

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. ♦

WIN NT

A user who attempts to log in to a Windows NT computer that is a member of a domain can either do so 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 only connect to an Informix database server if there is an account for the user ID in the Windows NT domain in which the database server runs. This rule also applies to trusted domains.* ♦

The .netrc File

The **.netrc** file is an optional file, located in the user's home directory, that specifies identity data. If a user is not a trusted user, or not on a computer that is trusted by the database server, you 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 use this file.

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

The database server uses the **.netrc** file regardless of whether it uses the default authentication policy or, on a UNIX platform, a communications support module.

For information about the specific content of this file, refer to your operating-system documentation.

WIN NT

On Windows NT, a home directory is not automatically assigned when the Windows NT 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 the client's behalf. 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 a **.netrc** file.

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

File or Statement	Example
.netrc file	`machine trngpc3 login bruce password im4golf`
CONNECT STATEMENT	`CONNECT TO ol_trngpc3 USER bruce USING "im4golf"`
informix.ini file	A client running on a Windows 3.1 computer that connects through INFORMIX-NET can use the **setnet** utility to store a user name and password in the **informix.ini** file.

Tip: *Windows 3.1 clients can also use the setnet utility to provide a password.*

UNIX

The $INFORMIXDIR/etc/concsm.cfg file

The **concsm.cfg** file describes the communications support module and is required only if you use the communications support module. An entry in the file is limited to 1024 characters. If the options parameter in the sqlhosts file does not designate the communications support module for the specified database server, the database server uses the default authentication policy.

The **concsm.cfg** file entry has the following format:

```
csmname("lib-path", "csm-global-option",
     "csm-connection-options")
```

The *csmname* variable is the name that you assign to the communications support module. The *lib-path* parameter is the full pathname, including the file name, of the shared library that is the communications support module. The *csm-global* option is not used at this time. The following section describes the *csm-connection* option for the DCE-GSS CSM.

The following example illustrates the parameter that you must enter in the **concsm.cfg** file to define the DCE-GSS communications support module.

```
dcegss("/usr/informix/lib/csm/libixdce.so")
```

This parameter assigns the name **dcegss** to the CSM and specifies that the shared-library pathname is **/usr/informix/lib/csm/libixdce.so**.

The concsm.cfg Connection Options Field for the DCE-GSS CSM

The CSM *connection-options* field for the DCE-GSS CSM has the following format:

```
letter = value
```

The following table describes the *csm-connection* option for the DCE-GSS CSM.

Option	Option name	Option Value	Default
confidentiality	c	1 or 0	0
dbserver principal name	p	string (no quotes)	*dbservername*
delegation	d	1 or 0 (zero)	0
integrity	i	1 or (zero)	0

For option names `c`, `d`, and `i`, a value of `1` enables the service and a value of `0` (zero) disables the service.

Use the `p` option if the DCE principal name of the database server is not the same as the **dbservername**.

If you want to specify a principal name for the database server that is different than the **dbservername**, you can specify the `p` *csm-connection* option, as in the following example:

```
dcegss("/usr/informix/lib/csm/libixdce.so", "", "p=myserver")
```

You can combine several options in the options string. The options must be separated by commas, and no spaces are allowed. You can include the options in any order. The following example enables confidentiality and integrity and specifies a principal name for the database server:

```
dcegss("/usr/informix/lib/csm/libixdce.so", "", "c=1,i=1,p=myserver")
```

For an overview of the communications support services, refer to "What Is a Connection?" on page 4-6. For the specific name of the CSM shared library, refer to the machine-notes file that is referenced in the section "Documentation Notes, Release Notes, Machine Notes" on page -15. For information on how to specify a communications support module in the **sqlhosts** file, refer to "The Communications Support Module Option" on page 4-45.

The sqlhosts File or Registry

Informix client / server connectivity information resides in the **sqlhosts** file or registry. The **sqlhosts** file or registry contains information that enables a client application to find and connect to any Informix database server on the network.

UNIX

The sqlhosts File

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 pathname and filename of a file that contains the **sqlhosts** file information.

You can enter information in the **sqlhosts** file by using a standard text editor.

Figure 4-4 shows a sample **sqlhosts** file.

Figure 4-4
Sample sqlhosts File

dbservername	nettype	hostname	servicename	options
menlo	onipcshm	valley	shm_file	
menlo2	ontlitcp	valley	menlo_on	
newyork	ontlitcp	hill	dynsrvr2	
pittsburgh	onsoctcp	canyon	1536	
sales	ontlispx	knight	sales	k=0,r=0
payroll	onsoctcp	dewar	py1	s=2,b=5120
personnel	ontlitcp	37.1.183.92	sales_ol	
texas_srvr	ontlitcp	*texas	pd1_on	

WIN NT

The sqlhosts Registry

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

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

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

Figure 4-5 illustrates the location and content of the **sqlhosts** registry for the sample database server **ol_nile**.

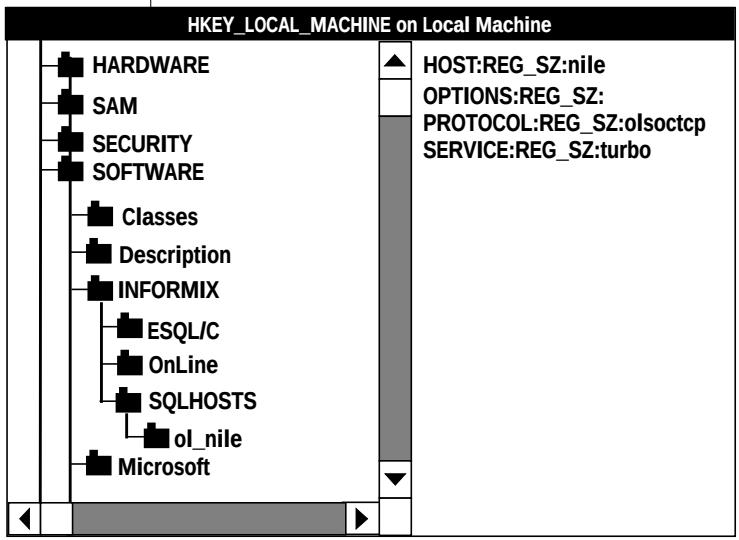


Figure 4-5
sqlhosts Information in the Windows NT Registry

When the client application runs on the same computer as the database server, they share a single **sqlhosts** registry.

The Location of the sqlhosts Registry

When you install the database server, you have the option of specifying the name of the computer where you want to store the **sqlhosts** registry. 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 relieves you of the necessity to maintain the same **sqlhosts** information on multiple computers.

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

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

Changing the sqlhosts Registry

You must use the Windows NT program **regedt32** to change an entry in the **sqlhosts** registry.

To change an entry

1. Run **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. Click the folder icon for the name of the database server.

5. Double-click the **sqlhosts** value that you want to edit (HOST, OPTIONS, and so on).
6. In the String Editor box, type the new value for the key that you selected and click **OK**.
7. Repeat steps 5 and 6 for each value that you want to change.
8. Exit the Registry Editor.

sqlhosts File and Registry Fields

The **sqlhosts** file (on UNIX) and the **sqlhosts** registry (on Windows NT) use the same fields to identify connectivity information; however, some of the field names are different.

sqlhosts File	sqlhosts Registry
dbservername	key name
nettype	protocol
hostname	host
servicename	service
options	options

The fields can be delimited by spaces or by tabs. You cannot include any spaces or tabs *within* a field. You can put comments in the **sqlhosts** file by starting a line with the comment character (#). Additional syntax rules for each of the fields are provided in the following sections, which describe the entries in the **sqlhosts** file or registry.

If you install INFORMIX-Enterprise Gateway with DRDA in the same directory as the database server, your **sqlhosts** file or registry will also contain 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 *INFORMIX-Enterprise Gateway with DRDA User Manual*.

The dbservername Field and Key Name Field

The **dbservername** or **key name** field provides 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 dbservername. The **dbservername** or **key name** 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. These configuration parameters are discussed in "ONCONFIG Parameters for Connectivity" on page 4-50. If the **sqlhosts** file or registry has multiple entries with the same dbservername, only the first one is used.

The database server looks up this information whenever you initialize the database server and when client applications attempt to connect to a database server.

The **dbservername** field can include any printable character other than an uppercase character, a field delimiter, a newline character, or a comment character. It is limited to 18 characters.

The nettype and Protocol Field

The **nettype** field describes the type of interface / protocol combination that should be made between the client application and the database server. The field is a series of eight letters composed of three subfields, as illustrated in Figure 4-6.

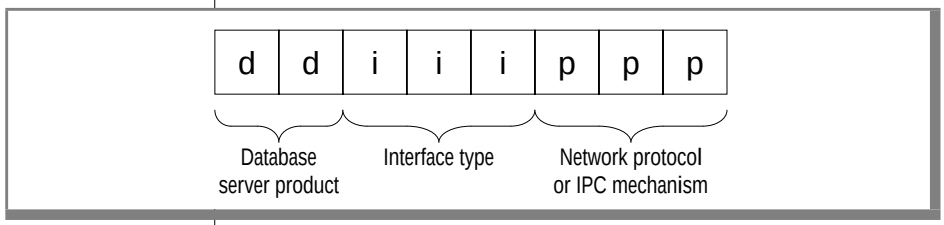


Figure 4-6
Format of the nettype Field

The subfields of **nettype** are as follows.

Database Server Name

The first two letters of **nettype** represent the database server product, as follows:

on Dynamic Server

ol Dynamic Server

dr INFORMIX-Enterprise Gateway with DRDA

Interface Type

The middle three letters of **nettype** represent the network programming interface that enables communications (see "What Is a Network Programming Interface?" on page 4-5).

Interface Subfield	Type of Interface
ipc	IPC (interprocess communications)
soc	Sockets
tli	TLI (transport layer interface)

UNIX

Interprocess communications (IPC) are UNIX-based connections used only for communications between two processes running on the same computer. For more information on supported network programming interfaces, see "What Interface / Protocol Combinations Are Available on Your Platform?" on page 4-13.

Network Protocol

The final three letters of **nettype** represent the network protocol or specific IPC mechanism.

Protocol Subfield	Type of Protocol
shm	Shared-memory communication
str	Stream-pipe communication ♦
nmp	Named pipe communication ♦
tcp	TCP/IP network protocol
spx	IPX/SPX network protocol

IPC connections use shared memory or, on UNIX, stream pipes. The database server supports two network protocols: TCP/IP and IPX/SPX. For more information on the protocols that the database server supports, see "What Connections Does the Database Server Support?" on page 4-8.

Figure 4-7 summarizes the **nettype** values for the database server. However, the database server might not support all these **nettype** values for your platform. For information on how to find out which **nettype** values the database server supports on your platform, see "What Interface/Protocol Combinations Are Available on Your Platform?" on page 4-13.

Figure 4-7
Summary of nettype Values

nettype	Description	Connection Type
onipcshm	The database server using shared-memory communication	IPC
onipcstr	The database server using stream-pipe communication	IPC ♦
onipcnmp	The database server using named-pipe communication	IPC ♦

(1 of 2)

nettype	Description	Connection Type
ontlitcp	The database server using TLI with TCP/IP protocol	Network ♦
onsoctcp	The database server using sockets with TCP/IP protocol	Network
ontlispx	The database server using TLI with IPX/SPX protocol	Network ♦

(2 of 2)

The hostname and host Field

The **hostname** or **host** field contains the name of the computer where the database server resides. The **hostname** or **host** fields can include any printable character other than a field delimiter, a newline character, or a comment character. The **hostname** or **host** field is limited to 256 characters.

The following sections explain how client applications derive the values used in the **hostname** field.

UNIX

Shared-Memory and Stream-Pipe Communication

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 runs.

Network Communication With TCP/IP

When you use the TCP/IP network protocol, the **hostname** field serves as a key to the **hosts** file, which provides the network address of the computer. The *hostname* that you use in the **sqlhosts** file must correspond to the *hostname* in the **hosts** file.

Using an IP Address with TCP/IP Connections

For TCP/IP connections (both TLI and sockets), you can use the actual Internet IP address in the **hostname** field instead of the host name or alias found in the **hosts** file. The IP address is always composed of four sets of one to three integers, separated by periods. Figure 4-8 shows a sample **hosts** file with IP addresses and host names. The host aliases are optional and can be omitted.

Internet IP Address	Host Name	Host Alias(es)
157.111.192.127	smoke	
49.192.4.63	odyssey	
37.1.183.92	knight	sales

Figure 4-8
A Sample hosts File

Using the IP address for `knight` from Figure 4-8, the following two **sqlhosts** entries are equivalent.

servername	nettype	hostname	servicename
sales	ontlitcp	37.1.183.92	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 **sqlhosts** file or registry makes it less convenient to identify the computer with which an entry is associated.

UNIX

You can find the IP address from the net address field of the **/etc/hosts** file or by using the UNIX **arp** or **ypmatch** command. ♦

WIN NT

You can configure Windows NT 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 with TCP/IP Connections

You can use wildcard addressing in the **hostname** field of the **sqlhosts** file or registry 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 **sqlhosts** file or registry used by the database server. When you enter an asterisk in the **hostname** 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, as in Figure 4-9 on page 4-37, the **hosts** file must have an entry for each interface card. For example, the **hosts** file for the **texas** computer might include these entries.

	Internet IP Address	Host Name
Card 1	123.34.6.81	texas1
Card 2	123.34.6.82	texas2

You can use the wildcard (*) alone or as a prefix for a host name or IP address, as shown in Figure 4-10 on page 4-38. The wildcard in the **hostname** field is meaningful only to the database server. If a client application uses an **sqlhosts** entry that contains a wildcard, the client application simply ignores the wildcard and searches for a host name after the wildcard.

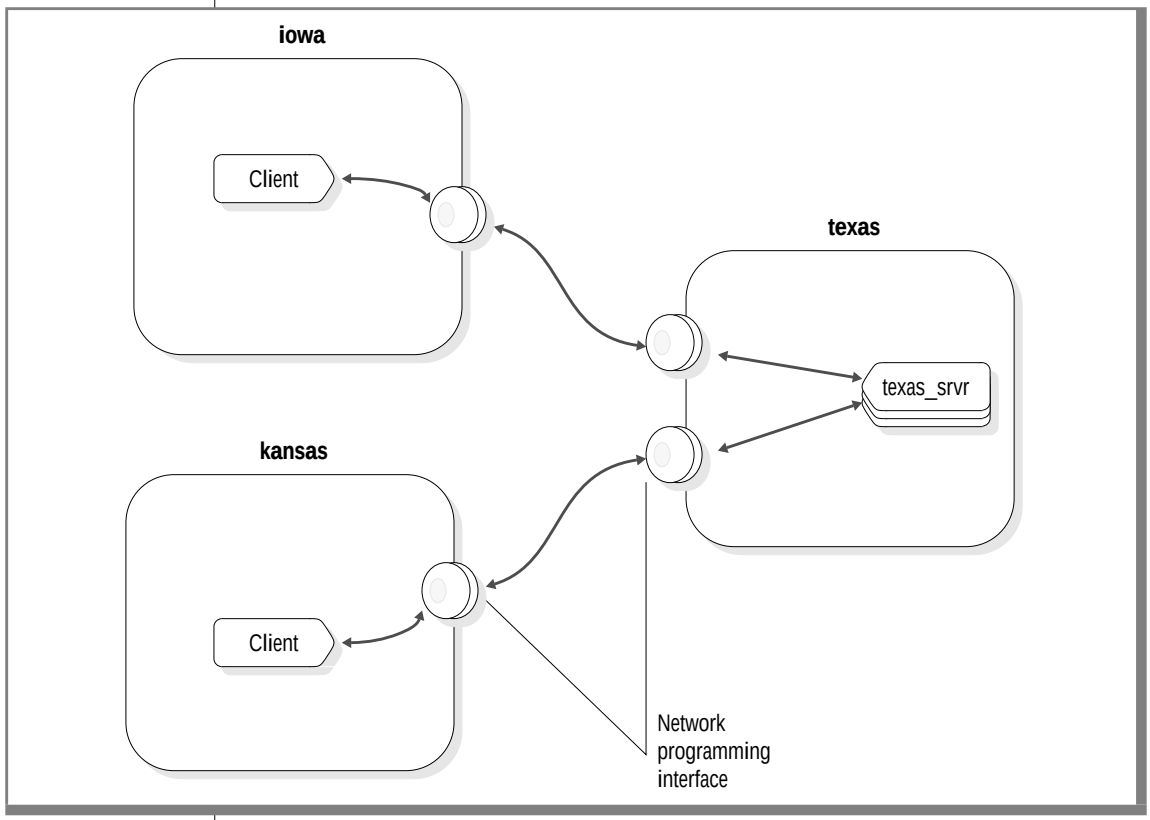
If the client application and database server share an **sqlhosts** file or registry, you can specify both the asterisk and a host name or IP address in the **hostname** field (for example, *texas1 or *123.34.6.81). The client application ignores the asterisk and uses the host name (or IP address) to 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 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.

Using Wildcard Addressing

Figure 4-9 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.

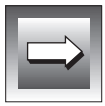
Figure 4-9
Using Multiple Network-Interface Cards



The **sqlhosts** file or registry for the **texas_srvr** database server can include any one of the entries in Figure 4-10.

Figure 4-10
Possible Entries sqlhosts for the texas_srvr Database Server

servername	nettype	hostname	servicename
texas_srvr	ontlitcp	*texas1	pd1_on
texas_srvr	ontlitcp	*123.34.6.81	pd1_on
texas_srvr	ontlitcp	*texas2	pd1_on
texas_srvr	ontlitcp	*123.34.6.82	pd1_on
texas_srvr	ontlitcp	*	pd1_on



Important: *You can include only one of these entries in your **sqlhosts** file or registry.*

If any of the preceding lines are in its **sqlhosts** file or registry, the **texas_srvr** database server can accept client connections from either of the network cards. The database server finds the wildcard in the **hostname** field and ignores the explicit host name.

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

The **sqlhosts** file used by a client application must contain an explicit host name or IP address. The client application on **iowa** can use any one of the entries shown in Figure 4-11 on page 4-39 in its **sqlhosts** file or registry.

Figure 4-11
Possible Entries in the sqlhosts File for the Client Application on iowa

servername	nettype	hostname	servicename
texas_srvr	ontlitcp	*texas1	pd1_on
texas_srvr	ontlitcp	*123.34.6.81	pd1_on
texas_srvr	ontlitcp	texas1	pd1_on
texas_srvr	ontlitcp	123.34.6.81	pd1_on

Important: *You can include only one of these entries in your* ***sqlhosts*** *file.*

The client application ignores the wildcard in the **hostname** field.

The client application on **kansas** can use any one of the entries shown in Figure 4-12 in its **sqlhosts** file or registry.

Figure 4-12
Possible Entries in the sqlhosts File for the Client Application on kansas

servername	nettype	hostname	servicename
texas_srvr	ontlitcp	*texas2	pd1_on
texas_srvr	ontlitcp	*123.34.6.82	pd1_on
texas_srvr	ontlitcp	texas2	pd1_on
texas_srvr	ontlitcp	123.34.6.82	pd1_on

Important: *You can include only one of these entries in your* ***sqlhosts*** *file or registry.*

UNIX

Network Communication with IPX/SPX

When you use the IPX/SPX network protocol, the **hostname** field of the **sqlhosts** file 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. However, in the* ***sqlhosts*** *file, you can enter the name in either uppercase or lowercase letters.*

The servicename and Service Field

The interpretation of the **servicename** field depends on the type of connection that the **nettype** field specifies.

The **servicename** and **service** fields can include any printable character other than a field delimiter, a newline character, or a comment character. The **servicename** and **service** fields are limited to 128 characters.

UNIX

Shared-Memory and Stream-Pipe Communication

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. Informix recommends that you use the **dbservername** as the **servicename** for stream-pipe connections.

WIN NT

Named-Pipe Communication

When the **nettype** field specifies a named-pipe connection (**onipcnmp**), the **servicename** entry contains the name that the user specified for the pipe, using the universal name convention (UNC) to specify the name. For example, if the user specified the name of the pipe as **somepipe**, the servicename entry would be as follows:

```
\\.PIPE\somepipe
```

The period following the double backward slashes means that the named-pipe connection is local.

Network Communication With TCP/IP

When you use the TCP/IP connection protocol, the **servicename** must correspond to a **servicename** 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 4-13 shows the relationship between the **sqlhosts** file or registry and the **hosts** file, as well as the relationship of **sqlhosts** to the services file.

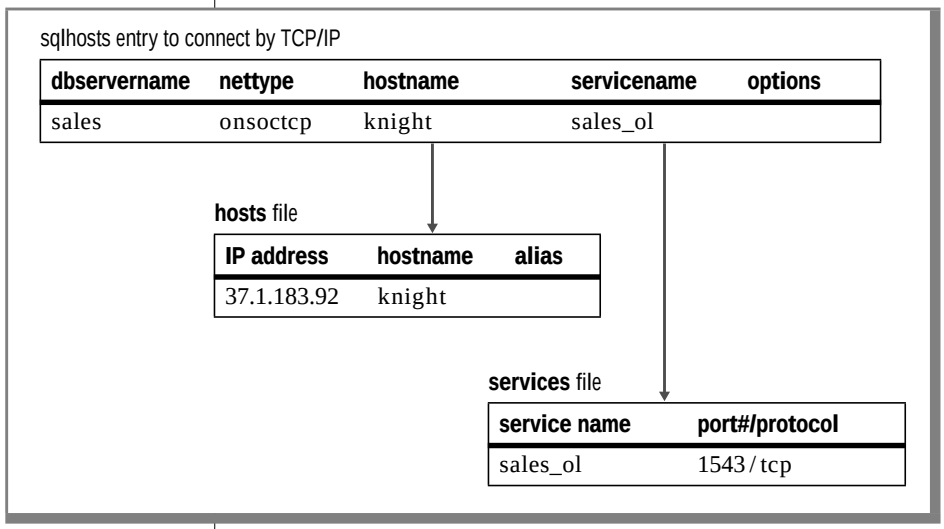


Figure 4-13
Relationship of sqlhosts File or Registry to hosts and services Files

Using the Port Number with TCP/IP Connections

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

Using the port number from the services file in Figure 4-13 on page 4-41, the **servicename** field of the **sqlhosts** entry could also be written as in the following example.

servername	nettype	hostname	servicename
sales	ontlitcp	knight	1536

Using the actual port number might save time when you make a connection in some circumstances. However, as with the IP address in the **hostname** field, using the actual port number might make administration of the **sqlhosts** file or registry less convenient.

UNIX

Network Communication With IPX/SPX

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.

The options Field

The **options** field of the **sqlhosts** file provides additional flexibility in specifying connections.

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

Option Name	Option Letter
Buffer size	b
Communication support module	csm ♦
End of group	e
Group	g
Identifier	i
Keep-alive	k
Security	s (database server) r (client)

UNIX

When you change the options in the **sqlhosts** file or registry, 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 only its own **sqlhosts** entry during initialization. If you change the options for the database server, you must reinitialize 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 characters.

Important: *The database server evaluates the option 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	Correct use of syntax
s=3,k=0 b=5120	Yes	Equivalent to the preceding entry (note the use of white space instead of a comma)
k=s=0	No	You cannot combine entries.

The Buffer-Size Option

Use the buffer-size option (`b=`) 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 database server default buffer size for TCP/IP is 4,096 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, refer to 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: *Informix recommends that you use the default size for the communications buffer. If you choose to set the buffer size to a different value, however, set the client-side communications buffer and the server-side communications buffer to the same size.*

UNIX

The Communications Support Module Option

Use the communications support module (CSM) option to describe the DCE-GSS CSM for each **sqlhosts** entry that you want to use it. If you do not specify the CSM option, the database server uses the default authentication policy for that dbservername. You can specify the same CSM option setting for all **sqlhosts** entries, or you can specify 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. For information on the **concsm.cfg** file entry, refer to "The $INFORMIXDIR/etc/concsm.cfg file" on page 4-25.

The following example illustrates how to specify a communications support module in the options field of the **sqlhosts** file.

```
csm=(dcegss,"p=my_server" )
```

This entry specifies that the **dcegss** communications support module will be used for the connection and that the principal name for the database server is **my_server**.

For more information on the communications support module, refer to "What Is a Connection?" on page 4-6. For more information on the **concsm.cfg** file, refer to "The $INFORMIXDIR/etc/concsm.cfg file" on page 4-25.

The End of Group Option

Use this option to specify the ending member of a database server group. You can use this option only in a group entry, but it is not a required option for each database server group entry. It you specify this option in an entry other than a database server group, it 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.

The Group Option

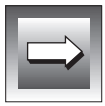
When you define database server groups in the **sqlhosts** file or registry, you can use multiple related **sqlhosts** entries as one logical entity to establish or change client/server connections. Enterprise Replication uses groups to declare database servers for replication. Use the following steps to help you create database server groups.

To name a database server group

1. Specify the name of the database server group to which the **sqlhosts** entry belongs (up to 18 characters) in the DBSERVERNAME field.

   The database server group name can be the same as the initial DBSERVERNAME for the database server.
2. The **nettype** field contains the keyword **group**.
3. Do not use the **hostname** and **servicename** fields when you initially name the database server group.

   If you specify options, these fields can contain dash (-) characters as null-field indicators for the unused fields. The only options available for a database server group entry are the end-of-group option (e) and the identifier option (i).



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

Figure 4-14 on page 4-47 shows an **sqlhosts** file or registry that defines database server groups.

Figure 4-14
Database Server Groups in sqlhosts File or Registry

dbservername	nettype	hostname	servicename	options
asia	group	–	–	e=asia.3
asia.1	ontlitcp	node6	svc8	g=asia
asia.2	onsoctcp	node0	svc1	g=asia
usa.2	ontlispx	node9	sv2	
asia.3	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

The example in Figure 4-14 shows the following two groups: **asia** and **peru**. Group **asia** includes the following members:

- asia.1
- asia.2
- usa.2
- asia.3

Because group **asia** uses the end-of-group option (e=asia.3), the database server searches for group members until it reaches **asia.3**, so the group includes **usa.2**.

Because group **peru** does not use the end-of-group option, the database server continues to include all members until it reaches the end of file.

Be careful of your placement of group members when you define database server groups.

The name of a database server group can also be used as a dbservername in the following environment variables:

- **INFORMIXSERVER**

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

- **DBPATH**

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

For more information on the use of the group option, refer to the *Guide to Informix Enterprise Replication*

The Identifier Option

The identifier option uses an identifier to specify a database server. The identifier must be a positive numeric integer and must be unique within **sqlhosts**.

For more information on the use of the identifier option, refer to the *Guide to Informix Enterprise Replication*.

The Keep-Alive Option

The keep-alive option is a network option that TCP / IP uses. It does not affect other types of connections.

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, the keep-alive option enables the network service to check 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 connection is considered broken, and all resources related to the connection are released.

When the keep-alive option is enabled, the network service spends additional resources to check the connection.

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.

If you do not include the keep-alive option in the **options** field, the keep-alive feature is enabled by default. You can set this option on the server side only, the client side only, or on both sides. For most cases, Informix recommends that you enable the keep-alive option.

The Security Options

The security options let you control operating-system security-file lookups. The letter *s* identifies 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 the **.netrc** lookup from the client side
r=1	Enables the **.netrc** lookup from the client side (default setting for the client side)
s=0	Disables both **hosts.equiv** and **.rhosts** lookup from the server side
s=1	Enables only the **hosts.equiv** lookup from the server side
s=2	Enables only the **.rhosts** lookup from the server side
s=3	Enables both **hosts.equiv** and **.rhosts** lookup on the server side (default setting for the server side)

The security options let you control the way that a client (user) gains access to a database server. By default, an Informix database server searches the clients **hosts.equiv** and **.rhosts** files to determine whether the client host computer is trusted. With the security options, you can specifically enable or disable the use of either or both the **hosts.equiv** and **.rhosts** files.

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

ONCONFIG Parameters for Connectivity

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

- DBSERVERNAME
- DBSERVERALIASES
- NETTYPE

The next sections explain the DBSERVERNAME and DBSERVERALIASES configuration parameters.

Although the NETTYPE parameter is not a required parameter, Informix recommends that you use it if you are going to configure two or more connection types. It lets you adjust the number and type of virtual processors used by the database server for communication. 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 DBSERVERNAME, DBSERVERALIASES, and NETTYPE, refer to "Network Virtual Processors" on page 9-28.

The 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 entry in the **sqlhosts** file or registry 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

The DBSERVERALIASES Configuration Parameter

The DBSERVERALIASES parameter lets you assign multiple dbservernames to the same database server. Figure 4-15 shows entries in an ONCONFIG configuration file that assign three dbservernames to the same database server instance.

Figure 4-15
Example of DBSERVERNAME and DBSERVERALIASES Parameters

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

The **sqlhosts** file or registry associated with the dbservernames from Figure 4-15 on page 4-51 could include the entries shown in Figure 4-16. Because each dbservername has a corresponding entry in the **sqlhosts** file or registry, you can associate multiple connection types with one database server.

Figure 4-16
Three Entries in the sqlhosts File for One Database Server

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

Using the **sqlhosts** file shown in Figure 4-16, 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_Usrvr'
```

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. These environment variables are used to minimize connection errors caused by busy network traffic. Environment variables are documented in the *Informix Guide to SQL: Reference*.

You do not need to set **INFORMIXCONTIME** or **INFORMIXCONRETRY** when you configure and initialize the database server. Users of client applications that connect to the database server using network connections might need to set these variables.

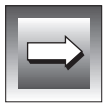
If the client application explicitly attaches to shared-memory segments, you might need to set **INFORMIXSHMBASE** (shared-memory base). For more information, refer to "Where the Client Attaches to the Communications Portion" on page 11-11.

The **INFORMIXSERVER** environment variable allows you to specify a default dbservername to which your clients will connect. For more information, see the *Informix Guide to SQL: Reference*.

Examples of Client/Server Configurations

The next several sections show the correct entries in the **sqlhosts** file or registry 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
- Using the relay module
- Using a post-6.0 client application with a Version 5.x database server

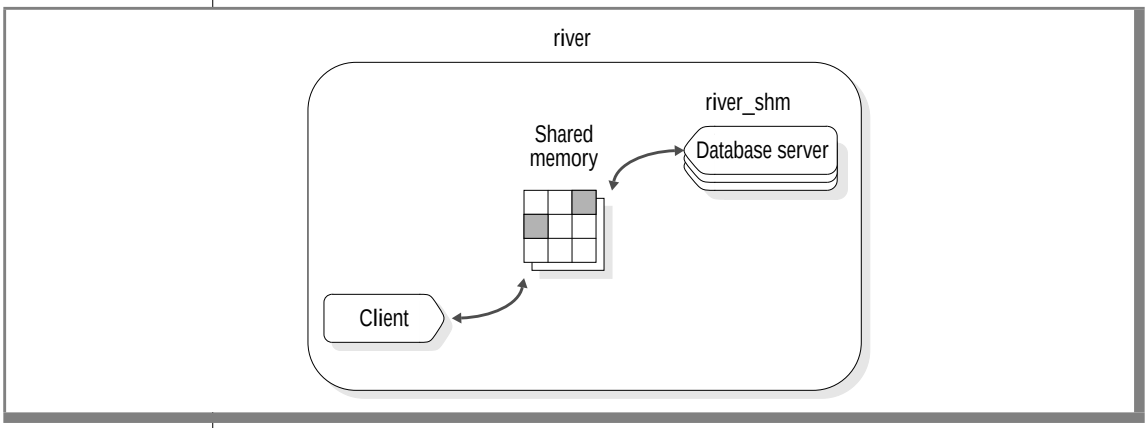


Important: *In the following examples, you can assume that the network-configuration files* ***hosts*** *and* ***services*** *have been correctly prepared, even if they are not explicitly mentioned.*

Using a Shared-Memory Connection

Figure 4-17 shows a shared-memory connection on the computer named **river**.

Figure 4-17
A Shared-Memory Connection



The ONCONFIG configuration file for this installation includes the following line:

```
DBSERVERNAME river_shm
```

A correct entry for the **sqlhosts** file or registry is as follows.

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 registry.

For more information about shared-memory connections, refer to "How a Client Attaches to the Communications Portion" on page 11-11.

Using a Local Loopback Connection

Figure 4-18 shows a local loopback connection. The name of the host computer is **river**.

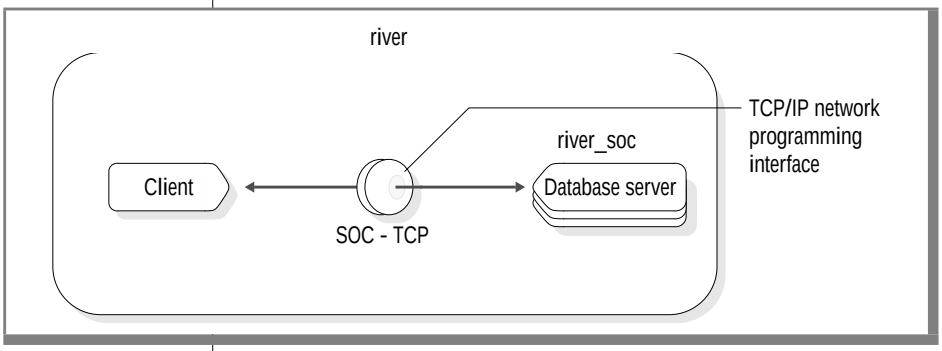


Figure 4-18
Local Loopback Connection

The network connection in Figure 4-18 uses sockets and TCP / IP, so the correct entry for the **sqlhosts** file or registry is as follows.

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:

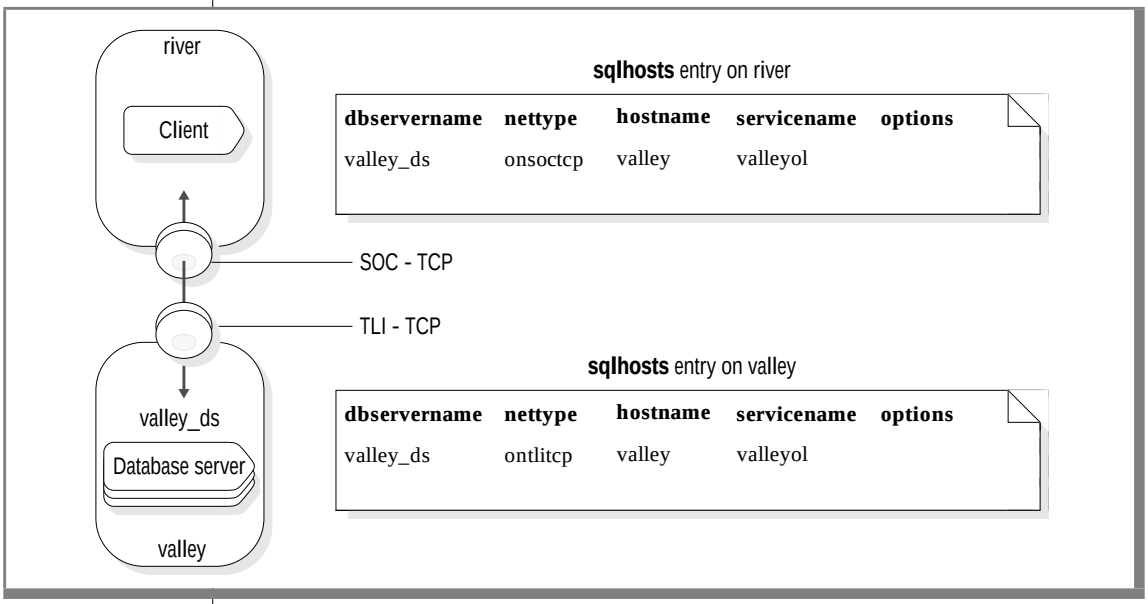
```
DBSERVERNAME river_soc
```

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 4-19 shows a configuration in which the client application resides on host **river** and the database server resides on host **valley**.

Figure 4-19
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 registry on the computer where the database server resides has a corresponding entry on the computer where the client application resides.

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`.

UNIX

The sqlhosts File Entry for IPX/SPX

IPX/SPX software frequently provides TLI. If the configuration in Figure 4-19 on page 4-56 uses IPX/SPX instead of TCP/IP, the entry in the **sqlhosts** file on both computers is as follows.

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 4-20 on page 4-58 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.

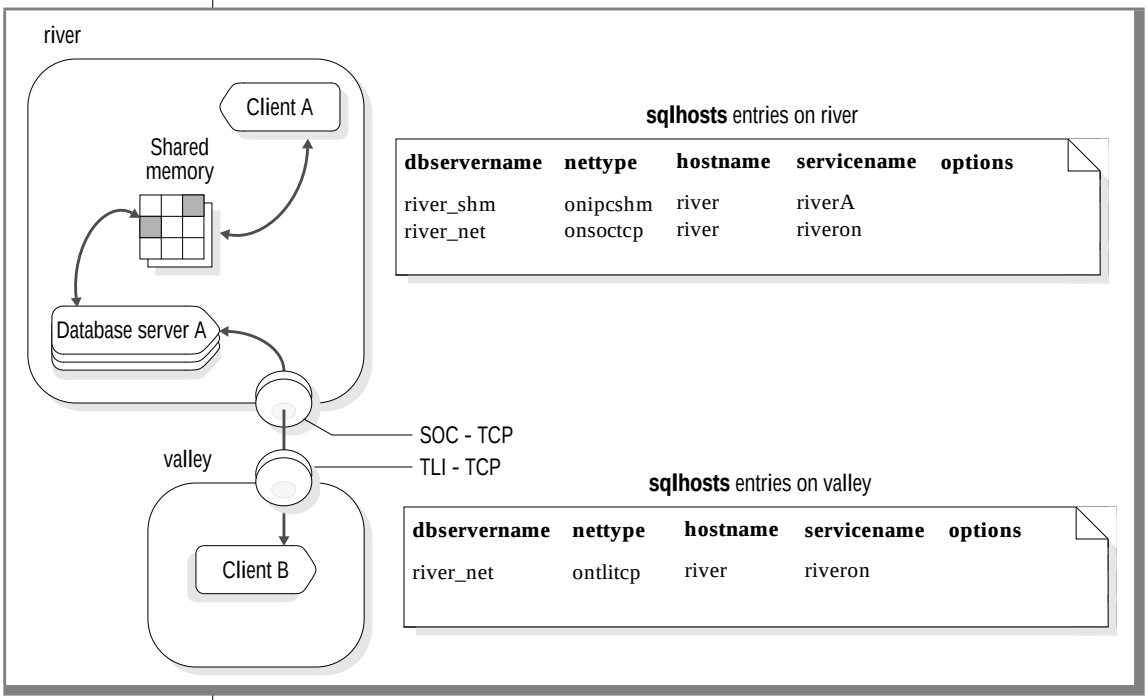
When you want the database server to accept more than one type of connection, you must take the following actions:

- Put DBSERVERNAME and DBSERVERALIASES entries in the ONCONFIG configuration file.
- Put an entry in the **sqlhosts** file or registry for each database server/connection type pair.

For the configuration in Figure 4-20 on page 4-58, 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 4-20
A Configuration That Uses Multiple Connection Types



The dbservername used by a client application determines the type of connection that is used. Client A connects to the database server using the following statement:

```
CONNECT TO '@river_shm'
```

In the **sqlhosts** file or registry, the **nettype** associated with the name **river_shm** specifies a shared-memory connection, so this connection is a shared-memory connection.

Client B connects to the database server using the following statement:

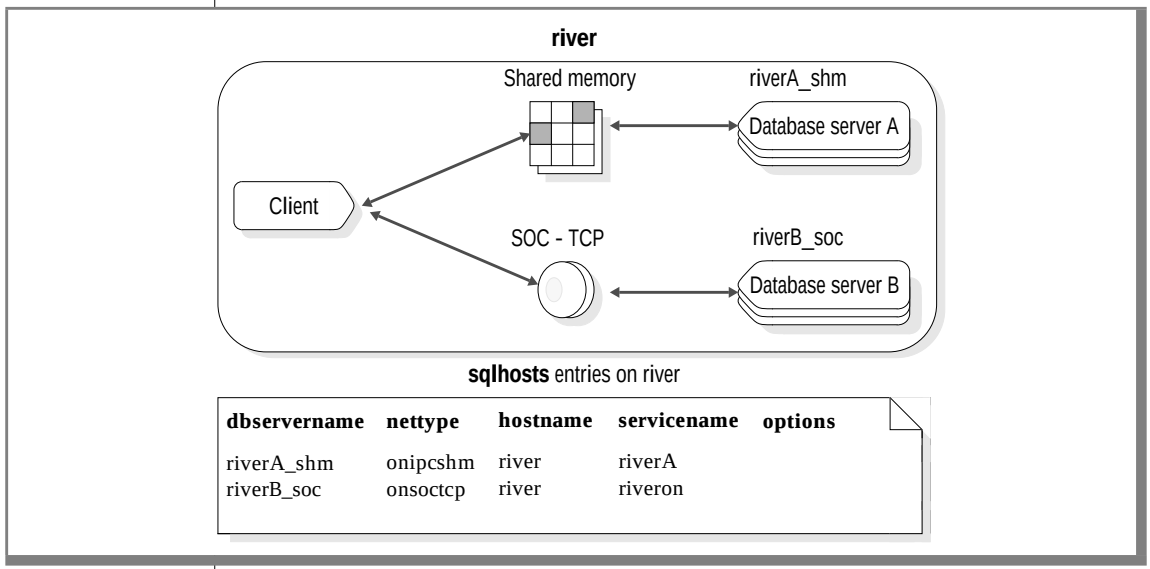
```
CONNECT TO '@river_net'
```

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 4-21 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 Chapter 5, "What Is Multiple Residency?")

Figure 4-21
Multiple Database Servers



sqlhosts entries on river

dbservername	nettype	hostname	servicename	options
riverA_shm	onipcshm	river	riverA	
riverB_soc	onsoctcp	river	riveron	

For the configuration in Figure 4-21, you must prepare two ONCONFIG configuration files, one for database server **A** and the other for database server **B**. The **sqlhosts** file or registry 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
```

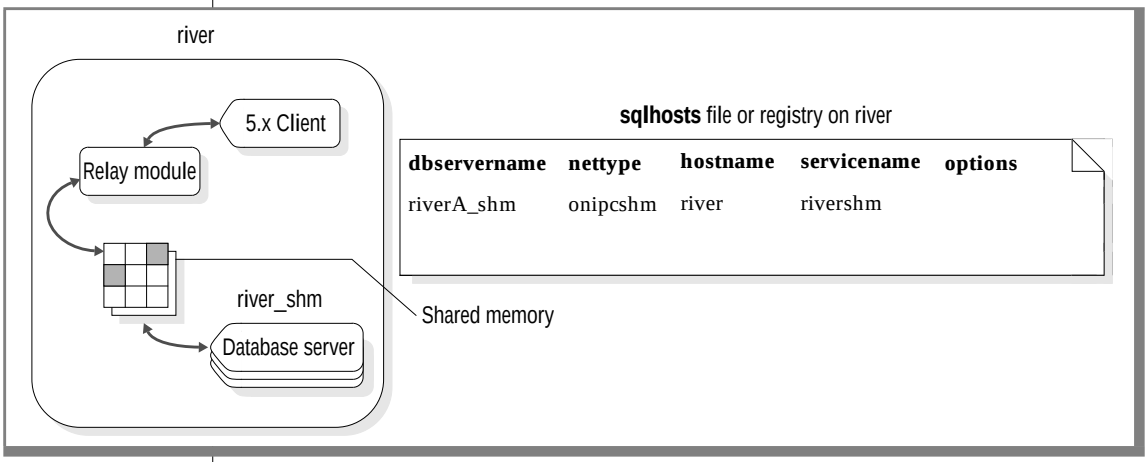
UNIX

Using the Relay Module

Starting with Version 6.0, each Informix database server includes a *relay module* that lets a Version 4.1 or Version 5.x client application connect to the database server. The connection can be either a network connection or a local connection that uses shared memory. The relay module lets Informix products from different release levels connect to each other as transparently as possible.

Figure 4-22 shows an example of a shared-memory connection between a Version 5.x client application and a post-6.0 database server using the relay module.

Figure 4-22
A Configuration with a 5.x Client Application and a post-6.0 Database Server



To prepare this configuration

1. Verify that a shared-memory connection between the database server and a client application of the same version works correctly. (In other words, prepare your configuration as you would for the configuration shown in Figure 4-17 on page 4-54.)

   The **sqlhosts** file does not contain an entry for the connection between the client and the relay module. The relay module does not affect the **sqlhosts** file.

2. The database server should be installed in the same directory as Version 5.x products. In this case, set the **SQLEXEC** environment variable to the pathname of the relay module, as shown in the following example. *(The relay module is stored as **$INFORMIXDIR/lib/sqlrm** as part of the installation process.*)

```
setenv SQLEXEC $INFORMIXDIR/lib/sqlrm
```

3. If you cannot install the database server in the same directory as the 5.x Informix products, do the following:

   a. Copy or symbolically link the subdirectories from the $**INFORMIXDIR/msg** directory of the database server product to the **$INFORMIXDIR/msg** directory of the Version 5.x products.

   b. Change the **INFORMIXDIR** environment variable to point to the directory where the Version 5.x products are installed.

   c. Modify the **PATH** environment variable to include **$INFORMIXDIR/bin** of Version 5.x database server.

   d. Set the **SQLEXEC** environment variable to the complete pathname of the relay module. You cannot use the variable **$INFORMIXDIR** to set the **SQLEXEC** environment variable, because the **INFORMIXDIR** environment variable now points to the directory of the Version 5.x products instead of to the directory where the database server products are stored. You must use the exact pathname, for example:

```
setenv SQLEXEC /usr/versionUSvr/informix/lib/sqlrm
```

   e. Remove extra environment variables.

      If Version 5.x Informix products are in use, the user's environment might include two environment variables that were required for Version 5.x database servers: **SQLRM** and **SQLRMDIR**. The user must unset these variables before the client application can use a post-6.0 database server, as the following example shows:

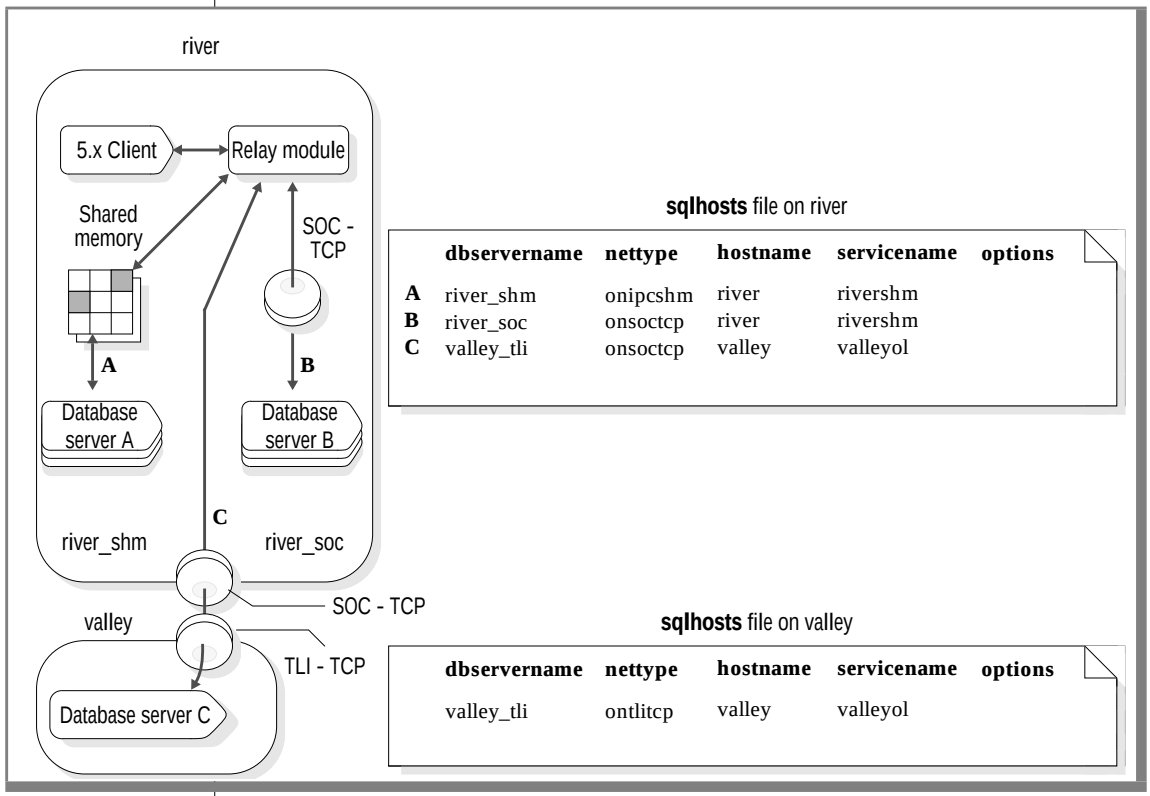
```
unsetenv SQLRM
unsetenv SQLRMDIR
```

The **DBNETTYPE** environment variable, used by the Version 5.x database servers, is not needed for later database servers. You can unset the **DBNETTYPE** environment variable if you want, but it does not affect post-6.0 products in any way.

A Relay Module Configuration with Three Database Servers

Figure 4-23 shows a configuration that has three possible connections (called A, B, and C) between a Version 5.x client application and three post-6.0 database servers. The client application can use any of the connections, but only one connection can be active at a time.

Figure 4-23
A Configuration with a 5.x Client, a Relay Module, and Three Database Servers



Verify that all three connections work correctly with a Version 5.x client application and then modify the environment variables.

UNIX

Using a Post-6.0 Client Application with a 5.x Database Server

To connect a post-6.0 client application to a Version 5.x database server, do the following:

- Use a network connection.
- Do not use any syntax that is specific to database servers after Version 5.x.

  For example, the client application cannot use the CONNECT statement because Version 5.x does not support the CONNECT statement.

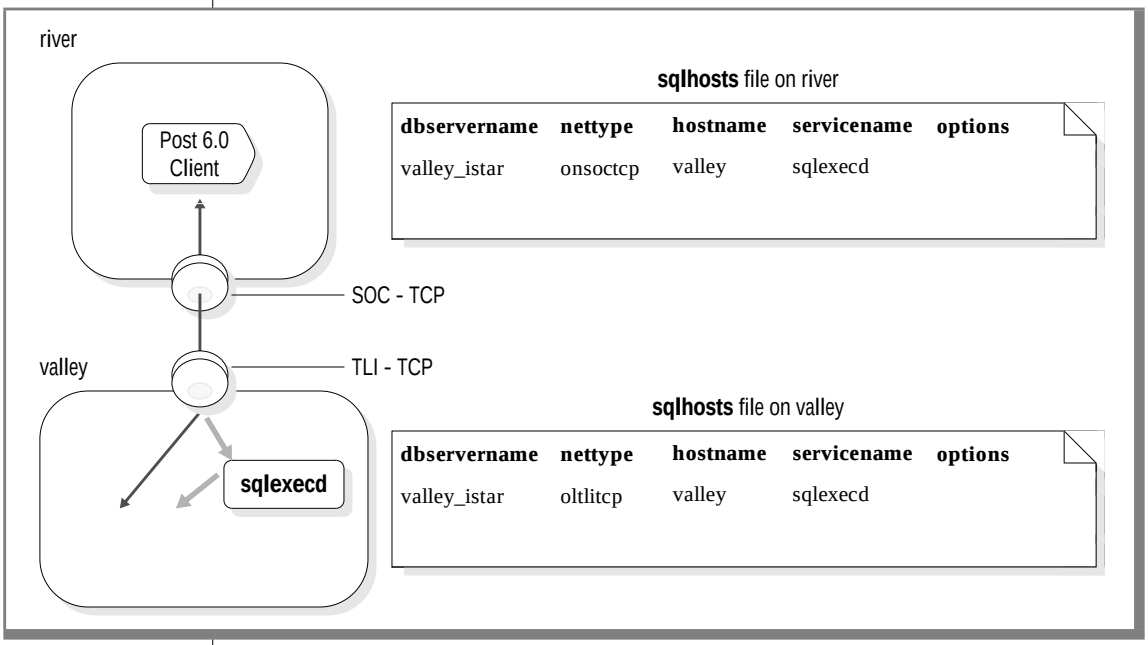
Important: *A post-6.0 client application is one that corresponds to an Informix database server that is Version 6.0 or later. An application that uses a product in the INFORMIX-Client Software Developer's Kit, Version 2.0 or later, is a post-6.0 client application.*

For example, to open a database named **5_base** using the database server **valley_istar**, do the following:

- Set the environment variable **INFORMIXSERVER** to **valley_istar**.
- In the application, issue the following statement:

```
DATABASE 5_base@valley_istar
```

Figure 4-24
Example of Connecting a Post-6.0 Client Application to a 5.x Database Server



To prepare to use the configuration in Figure 4-24 on page 4-64, follow these steps:

1. On the server host computer (**valley**), start the Version 5.x database server.
2. To confirm that the database server is active, connect a Version 5.x client application to the database server.
3. Write down (for use in step 6) the entry that is in the **sqlhosts** file for the Version 5.x database server.
4. On the client host computer (**river**), set the **INFORMIXDIR** environment variable to the directory where the database server products are installed.
5. Modify the **PATH** environment variable to include **$INFORMIXDIR/bin**.
6. Update the **sqlhosts** file to contain an entry for the Version 5.x database server.
7. Set the **INFORMIXSERVER** environment variable to the dbservername of the Version 5.x database server instance.

Chapter

5

What Is Multiple Residency?

Benefits of Multiple Residency 5-3

How Multiple Residency Works 5-4
The Role of the ONCONFIG Environment Variable 5-5
The Role of the SERVERNUM Configuration Parameter. 5-5

You can use more than one database server in the following two ways:

- Run multiple instances of the database server on a single host computer
- Access several database servers over a network

When multiple database servers and their associated shared memory and disk structures coexist on a single computer, it is called *multiple residency*. This chapter covers the concepts of multiple residency.

Benefits of Multiple Residency

Creating independent database server environments on the same computer allows you to perform the following actions:

- Separate production and development environments
- Isolate sensitive databases
- Test distributed data transactions on a single computer

When you use multiple residency, each database server has its own configuration file. Thus, you can create a configuration file for each database server that meets its special requirements for backups, shared-memory use, and tuning priorities.

You can separate production and development environments to protect the production system from the unpredictable nature of the development environment. You might also find it useful to isolate applications or databases that are critically important, either for security reasons or to accommodate more frequent backups than the majority of the databases require.

If you are developing an application for use on a network, you can use local loopback (see "Using a Local Loopback Connection" on page 4-55) to perform your distributed-data simulation and testing on a single computer. Later, when a network is ready, you can use the application without changes to application source code.

However, running multiple database servers on the same computer is not as efficient as running one database server. You need to balance the advantages of separate database servers against the extra performance cost.

How Multiple Residency Works

Multiple residency is possible because the operating system can maintain separate areas in shared memory and disk for each instance of the database server. Each instance of the database server passes a value to the operating system. This value, which is a function of the SERVERNUM parameter, specifies the shared-memory address to which the database server process should attach. You must also specify a unique database server name and unique storage locations for each instance of the database server.

The Role of the ONCONFIG Environment Variable

Each instance of the database server is described by the parameters in an ONCONFIG configuration file. The **ONCONFIG** environment variable specifies the name of the current ONCONFIG configuration file. The following configuration parameters should have unique values for each database server:

- SERVERNUM
- ROOTPATH and / or ROOTOFFSET
- DBSERVERNAME and DBSERVERALIASES
- MSGPATH
- MIRRORPATH and / or MIRROROFFSET

How to set these parameters is discussed in Chapter 6, "Using Multiple Residency."

The Role of the SERVERNUM Configuration Parameter

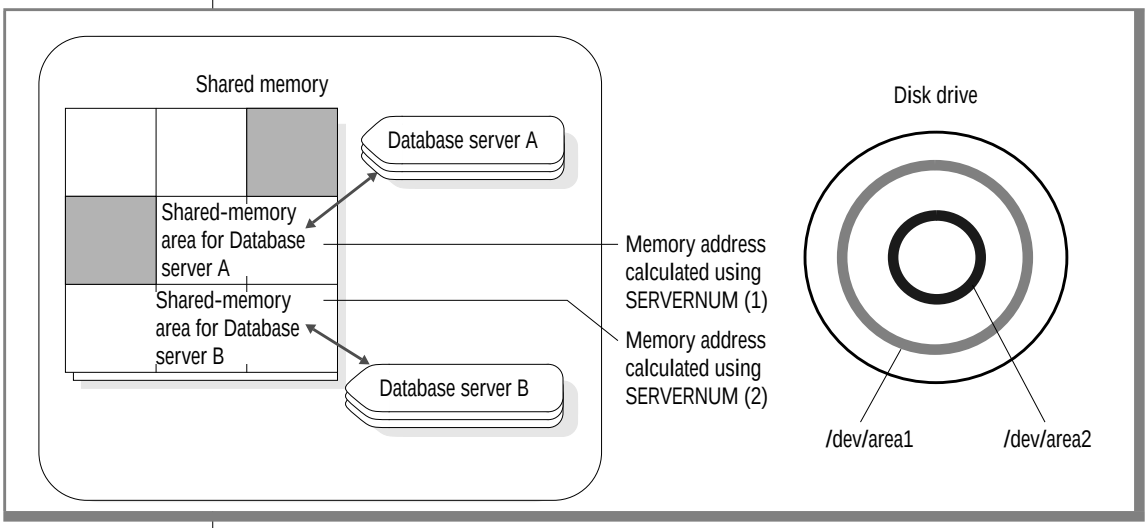
To maintain separation between the instances of database servers, you maintain multiple configuration files, each with a unique SERVERNUM value. When you initialize the database server, it reads the **ONCONFIG** environment variable for the name of its configuration file. Next, the database server reads its configuration file to obtain the value of its SERVERNUM parameter. The database server then uses the SERVERNUM value to calculate the required shared-memory address.

For example, the ONCONFIG files for two database servers on a UNIX platform might include these parameters.

ONCONFIG file: onconfig.one	ONCONFIG file: onconfig.two
.	.
.	.
DBSERVERNAME dbsrvr_one	DBSERVERNAME dbsrvr_two
SERVERNUM 1	SERVERNUM 2
ROOTPATH /dev/area1	ROOTPATH /dev/area2
.	.
.	.

Figure 5-1 provides an example of multiple residency using the configuration files shown in the preceding table. Each database server has its own name, its own section of shared memory, and its own storage area on disk.

Figure 5-1
Separate Memory and Storage in Multiple Residency



Chapter

6

Using Multiple Residency

Planning for Multiple Residency 6-3

Preparing for Multiple Residency 6-4
- Prepare a Configuration File 6-5
- Set the ONCONFIG Environment Variable 6-5
- Edit the New Configuration File 6-5
- Add Connection Information 6-7
- Update the sqlhosts File or Registry. 6-7
- Initialize Disk Space 6-7
- Prepare Dbspace and Logical-Log Backup Environment. 6-8
  - If You Use ON-Archive. 6-8
- Update the Operating-System Boot File 6-9
- Check Users' INFORMIXSERVER Environment Variables 6-9

Using instmgr.exe 6-10
- Adding an Instance of the Database Server 6-10
- Deleting an Instance of the Database Server 6-11

This chapter describes how to use multiple database servers on the same computer. It includes the following topics:

- Questions that you should ask when you plan for multiple residency
- Steps that you should follow for multiple residency

Before you perform this procedure, you should already have installed one database server as described in Chapter 3, "Configuring the Database Server."

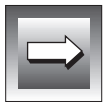
Planning for Multiple Residency

Each database server must have its own unique storage space. You cannot use the same disk space for more than one instance of a database server. When you prepare an additional database server, you need to repeat some of the planning that you did for installing the first database server. For example, you need to consider these questions:

- Will you use buffered or unbuffered files? Will the unbuffered files share a disk partition with another application? (For more information on buffered and unbuffered files, see "Unbuffered Disk Management" on page 1-6.)
- Will you use mirroring? Where will the mirrors reside?
- Where will the message log reside?
- Can you dedicate a tape drive to this database server for its logical logs?
- What kind of backups will you perform?

Preparing for Multiple Residency

The following sections present the steps to setup an additional database server, or multiple residency, on your computer.



WIN NT

Important: *Do not try to install another copy of the database server binaries. All instances of the same version of the database server on one computer share the same executable files.*

If you have installed Informix Enterprise Command Center (IECC) on a Windows NT computer, you can run **instmgr.exe** from the command line to create automatically an additional instance of the database server on that computer. The **instrmgr.exe** program uses a graphical interface to prompt you for the information that it needs. For information on how to run **instmgr.exe** to create an additional database server, refer to "Using instmgr.exe" on page 6-10. ♦

If you are not using IECC, you must perform the following steps manually to create an additional instance of the database server.

To create a multiple residency of a database server

1. Prepare a new ONCONFIG configuration file.
2. Set your **ONCONFIG** environment variable to the new filename.
3. Edit the new ONCONFIG configuration file.
4. If needed, add a servicename to the **services** file or connection information to the NetWare server.
5. Update the **sqlhosts** file or registry to include the dbservername(s) of the new database server.
6. Initialize disk space for the new database server.
7. Prepare dbspace and backup schedules.
8. Modify the operating-system boot file.
9. Check users' **INFORMIXSERVER** environment variable.

The sections that follow describe each of these steps.

Prepare a Configuration File

Each instance of the database server must have its own ONCONFIG configuration file. To prepare an ONCONFIG file for the new instance of a database server, copy a configuration file that already exists and modify it appropriately. You can copy a configuration that you have already prepared, or you can copy the **onconfig.std** file. *Do not modify **onconfig.std**.*

The following table lists the directories in which the configuration files must be located.

Platform	Directory
UNIX	**$INFORMIXDIR/etc**
Windows NT	**%INFORMIXDIR%\etc**

Give the new ONCONFIG file a name that you can easily associate with its function and its SERVERNUM value. For example, you might select the filename **onconfig.dev37** to indicate the configuration file for a development environment with the SERVERNUM value of 37.

Set the ONCONFIG Environment Variable

Set the **ONCONFIG** environment variable to the filename of the new ONCONFIG file. Specify only the filename, not the complete path.

Edit the New Configuration File

You can edit the new ONCONFIG file using a text editor or ON-Monitor.

Warning: *If you use ON-Monitor, you must set the **ONCONFIG** environment variable to the name of the new configuration file, and you must change SERVERNUM in the file (using a text editor) before you enter ON-Monitor. If you do not, you will edit the values of the wrong configuration file.*

In the new configuration file, you must change the following configuration parameters:

- SERVERNUM

  The SERVERNUM parameter specifies an integer (between 0 and 255) associated with a database server configuration. Each instance of a database server on the same host computer must have a unique SERVERNUM value. For more information, refer to "The Role of the SERVERNUM Configuration Parameter" on page 5-5.

- DBSERVERNAME

  The DBSERVERNAME parameter specifies the dbservername of a database server. Informix suggests that you choose a name that gives information about the database server, such as **ondev37** or ***hostname*dev37**. For more information, refer to "The DBSERVERNAME Configuration Parameter" on page 4-51.

- MSGPATH

  The MSGPATH parameter specifies the pathname of the message file for a database server. You should specify a unique pathname for the message file because database server messages do not include the dbservername. If multiple database servers use the same MSGPATH, you cannot identify the messages from separate database server instances. For example, if you name your database server **ondev37**, you might specify **/usr/informix/dev37.log** as the message log for this instance of the database server.

- ROOTPATH and / or ROOTOFFSET

  The ROOTPATH and ROOTOFFSET parameters together specify the location of the root dbspace for a database server. The root dbspace location must be unique for every database server configuration.

  If you put several root dbspaces in the same partition, you can use the same value for ROOTPATH. However, in that case, you must set ROOTOFFSET so that the combined values of ROOTSIZE and the ROOTOFFSET define a unique portion of the partition. For more information, refer to "ROOTPATH" on page 33-77 and "ROOTOFFSET" on page 33-76.

Tip: *You do not need to change ROOTNAME. Even if both database servers have the name* ***rootdbs*** *for their root dbspace, the dbspaces are unique because ROOTPATH and ROOTOFFSET specify a unique location.*

You might also need to set the MIRRORPATH and/or the MIRROROFFSET parameter. If the root dbspace is mirrored, the location of the root dbspace mirror must be unique. For information about setting MIRRORPATH refer to "Steps Required for Mirroring Data" on page 24-3.

Add Connection Information

If you use the TCP/IP communication protocol, you might need to add an entry to the **services** file for the new database server instance. If you use the IPX/SPX communication protocol, you might need to modify the connection information for the NetWare server.

Update the sqlhosts File or Registry

The **sqlhosts** file or registry must have an entry for each database server. If Informix products on other computers access this instance of the database server, the administrators on those computers must update their **sqlhosts** files or registry. Chapter 4, "Client/Server Communications," discusses the preparation of the **sqlhosts** file or registry.

If you plan to use TCP/IP network connections with an instance of a database server, the system network administrator must update the **hosts** and **services** files. If you use an IPX/SPX network, the NetWare administrator must update the NetWare file-server information. For information about these files, refer to "Network-Configuration Files" on page 4-17 and "IPX/SPX Connectivity Files" on page 4-22.

Initialize Disk Space

Before you initialize disk space, check the setting of your **ONCONFIG** environment variable. If you have not set it correctly, you might overwrite data from another database server. When you initialize disk space for a database server, the database server initializes the disk space specified in the current ONCONFIG configuration file.

Warning: *As you create new blobspaces or dbspaces for a database server, be sure you assign each chunk to a unique location on the device. The database server does not allow you to assign more than one chunk to the same location within a single database server environment, but it remains your responsibility as administrator to make sure that chunks belonging to different database servers do not overwrite each other.*

Prepare Dbspace and Logical-Log Backup Environment

This section provides a brief discussion of the effects of multiple residency on backups.

When you use multiple residency, you must maintain separate dbspace and logical-log backups for each database server instance. When you perform dbspace and logical log backups with multiple residency, you need to be especially aware of the following points:

- Device use
- Cataloger processes, if you use ON-Archive
- The **config.arc** file, if you use ON-Archive
- The **oper_deflt.arc** file, if you use ON-Archive

If you can dedicate a tape drive to each database server, you can back up your logical-log files using the continuous logging option. Otherwise, you must plan your dbspace and logical-log backup schedules carefully, so that use of a device for one database server instance does not cause the other database server instance to wait. You must reset the ONCONFIG parameter each time that you switch your backup operations from one database server instance to the other.

UNIX

If You Use ON-Archive

If you use ON-Archive for your backup and restore tool, each database server instance is served by its own **oncatlgr** process. When you start and stop **oncatlgr** processes using the startup and shutdown scripts (**start_oncatlgr** and **stop_oncatlgr**), the script prompts you to kill existing **oncatlgr** processes. You need to know which **oncatlgr** process serves the different database server instances before you kill them. Your *Archive and Backup Guide* explains how to associate the process ID of an **oncatlgr** process with a database server instance.

You can direct a database server to use a personalized copy of **config.arc** and **oper_deflt.arc**. To do this, set the **ARC_CONFIG** and **ARC_DEFAULT** environment variables to the names of your personalized configuration files. For example, suppose you copy **config.arc** and **oper_deflt.arc** to **config.mine** and **oper_deflt.mine**, respectively. You can then edit both of the **.mine** files to suit your needs. To direct a database server to use these files as your configuration files, set **ARC_CONFIG** to **config.mine** and **ARC_DEFAULT** to **oper_deflt.mine.**

UNIX

Update the Operating-System Boot File

You can ask your system administrator to modify the system startup script ("Prepare Operating-System Startup and Shutdown Scripts" on page 3-31) so that each of your database server instances starts whenever the computer is rebooted (for example, after a power failure).

The startup script for a single database server instance should set the **INFORMIXDIR**, **PATH**, **ONCONFIG**, and **INFORMIXSERVER** environment variables and then execute **oninit**. To start a second instance of a database server, change the **ONCONFIG** and **INFORMIXSERVER** environment variables to point to the configuration file for the second database server, and then execute **oninit** again. Do not change **INFORMIXDIR** or **PATH**.

Similarly, you can ask the system administrator to modify the shutdown script so that all instances of a database server shut down in a graceful manner.

UNIX

Check Users' INFORMIXSERVER Environment Variables

If a new instance of a database should be the default database server, your users need to reset the **INFORMIXSERVER** environment variable. Your users might need to update their **.informix** files.

If you use the **informix.rc** file to set environment variables for the users, you might need to update that file. The *Informix Guide to SQL: Reference* describes the **informix.rc** and **.informix** files.

WIN NT

Using instmgr.exe

On the Windows NT platform, you can run **instmgr.exe** to add or delete an instance of a version 7.3 database server. You must be a member of the **Informix-Admin** group to execute **instmgr.exe**.

Important: *On Windows NT, if you install multiple instances of the database server on one computer, all instances must be Version 7.3. Earlier versions of the database server on Windows NT do not support multiple residency.*

You can execute **instmgr.exe** in one of the following ways:

- Choose **Start→Programs→Informix Administration Tools group.**
- Locating **instmgr.exe** with Windows NT Explorer and click it.

  You can find **instmgr.exe** in the **%INFORMIXDIR%\bin** directory.
- Enter `instmgr` on an MS-DOS command line.

The **instmgr.exe** program displays a graphical interface that displays the names of the database servers that are already installed and gives you the option to create an additional instance of the database server or delete an instance that already exists.

For information about installing a database server initially and for additional information about adding an instance of the database server, refer to the *Installation Guide for Informix Dynamic Server on Windows NT*.

Adding an Instance of the Database Server

When you add an instance of the database server, you have the option to select one of the following types of installations:

- Typical
- Compact
- Custom

The Typical option is the default selection.

When you click the **Create** button, the **instmgr.exe** program prompts you to enter values for Server Number, Server Name, Service Name, Port Number, the Primary Storage Location, the Password of the system administrator, and name of the computer to use for the Shared Server.

The Server Number and Server Name values that you enter set the values for the SERVERNUM and DBSERVERNAME configuration parameters, respectively.

The Service Name and Port become the servicename and port entries for the server in the **sqlhosts** registry.

The Primary Storage Location specifies the location of the ROOTPATH configuration parameter.

The Password entry ensures that you have administrator privileges to add the new instance.

The name of the Shared Server computer specifies where to update the **sqlhosts** registry information for the new database server instance.

After you have entered the required information, the **instmgr.exe** program proceeds to install services, record environment variables, update the **sqlhosts** registry, and update the ONCONFIG file for the new database server instance.

For more information about the operations that the **instmgr.exe** program performs, refer to "Preparing for Multiple Residency" on page 6-4, which describes the steps that you would need to perform manually to create a new instance of the database server.

Deleting an Instance of the Database Server

To delete an instance of the database server, select a database server instance from the list of server instances and click the **Delete** button.

The **instmgr.exe** program first checks to see if on-line services for the database server are running and, if so, stops them. Then it removes the files and directories, icons, services, registry keys, dbspaces, and configuration information for the database server.

Modes and Initialization

Section III

Chapter

7

Managing Database Server Operating Modes

Database Server Operating Modes 7-3

Changing Database Server Operating Modes 7-5
- From Off-Line to Quiescent 7-5
- From Off-Line to On-Line 7-6
- From Quiescent to On-Line. 7-6
- Gracefully from On-Line to Quiescent 7-7
- Immediately from On-Line to Quiescent 7-7
- From Any Mode Immediately to Off-Line 7-8

This chapter introduces you to the database server operating modes and provides instructions on how to change the operating modes with command-line utilities.

UNIX

On UNIX, you can also use ON-Monitor to display and manipulate the database server operating modes. ♦

If you are using Informix Enterprise Command Center (IECC) to administer the database server, refer to the *Informix Enterprise Command Center User Guide* for information on how to determine and change the database server operating modes.

Database Server Operating Modes

You can determine the current database server mode by executing the **onstat** utility from the command line. The mode is displayed in the header.

The database server has three principal modes of operation, as Figure 7-1 on page 7-4 illustrates.

Figure 7-1
Operating Modes

Operating Mode	Description
Off-line mode	When the database server is not running. No shared memory is allocated.
Quiescent, or Administration, mode	When the **oninit** processes are running, shared-memory resources are allocated, but the system does not allow database user access. Only the administrator (user **informix**) can access the database server.
On-line mode	Users can connect with the database server and perform all database activities. This is the normal operating mode of the database server. User **informix** or user **root** can use the command-line utilities to change many database server ONCONFIG parameter values while the database server is on-line.

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 high-availability data-replication pair. An application can query a 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 a system restore. Recovery occurs during the change from Off-Line to Quiescent mode.
- *Shutdown mode* is transitory. It occurs when the database server is moving from on-line to quiescent mode or from on-line (or quiescent) to off-line mode. Current users access the system, but no new users are allowed access.

  Once 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. Each section includes information on how to change operating modes with the **oninit** and **onmode** utilities and with ON-Monitor. To change database server operating modes with IECC, refer to the *Informix Enterprise Command Center User Guide*.

Important: *Only users who are logged in as either* ***root*** *or* ***informix*** *can perform database server mode changes.*

From Off-Line to Quiescent

When the database server changes from off-line mode to quiescent mode, the database server initializes shared memory.

When the database server is in quiescent mode, no sessions can gain access to the database server. In quiescent mode, any user can see status information, and user **informix** or user **root** can access administrative options.

Quiescent mode is the same as Administration mode when you use IECC.

Utility	Action
oninit	■ Execute **oninit -s**. ■ Execute **onstat**. ■ Check the **onstat** header information for the current database server operating mode.
UNIX ON-Monitor	Select **Mode→Startup**.

From Off-Line to On-Line

When you take the database server from off-line mode to on-line mode, the database server initializes shared memory. When the database server is in on-line mode, it is accessible to all database server sessions.

Utility	Action
oninit	■ Execute **oninit**. ■ Execute **onstat**. ■ Check the **onstat** header information for the current database server operating mode.
ON-Monitor	Not supported

UNIX

From Quiescent to On-Line

When you take the database server from quiescent mode to on-line mode, all sessions gain access.

If you have already taken the database server from on-line mode to quiescent mode and you are now returning the database server to on-line mode, any users who were interrupted in earlier processing must reselect their database and redeclare their cursors.

Utility	Action
onmode	■ Execute **onmode -m**. ■ Execute **onstat**. ■ Check the **onstat** header information for the current database server operating mode.
ON-Monitor	Select **Mode→On-Line**.

UNIX

Gracefully from On-Line to Quiescent

Take the database server gracefully from on-line 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.

Once you initiate the mode change, it cannot be cancelled. During the mode change from on-line to quiescent, the database server is considered to be in Shutdown mode.

Utility	Action
onmode	■ Execute **onmode -m** or **onmode -sy** from the command line. ■ Execute **onstat**. ■ Check the **onstat** header information for the current database server operating mode.
UNIX ON-Monitor	Select **Mode→Graceful Shutdown**. ON-Monitor displays a list of all active user threads and updates it every five seconds until the last user thread completes work.

Immediately from On-Line to Quiescent

Take the database server immediately from on-line 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 performs proper cleanup on behalf of all sessions that were terminated by the database server. Active transactions are rolled back.

Utility	Action
onmode	■ Execute **onmode -u** or **onmode -uy** from the command line. ■ Execute **onstat**. ■ Check the **onstat** header information for the current database server operating mode.
UNIX ON-Monitor	Select **Mode→Immediate Shutdown**. ON-Monitor displays a list of all active user threads and updates it every five seconds until the last user thread completes work or until you leave the screen.

From Any Mode Immediately to Off-Line

You can take the database server immediately from any mode to off-line mode. A prompt asks for confirmation to go off-line. 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 off-line mode, reinitialize shared memory by taking the database server to quiescent or on-line mode. When you reinitialize shared memory, the database server performs a fast recovery to ensure that the data is logically consistent.

The database server performs proper cleanup on behalf of all sessions that were terminated by the database server. Active transactions are rolled back.

Utility	Action
onmode	Execute **onmode -k** or **onmode -ky** from the command line. The **-y** option eliminates an automatic prompt to confirm an immediate shutdown.
ON-Monitor	Select **Mode→Take Offline**. ON-Monitor displays a list of all active user threads and updates it every five seconds until the last user thread completes work or until you leave the screen.

UNIX

Chapter

8

Initializing the Database Server

Types of Initialization 8-3

Initialization Commands 8-4

Initialization Steps 8-4
- Process Configuration File 8-6
- Create Shared-Memory Portions 8-7
- Initialize Shared-Memory Structures 8-8
- Initialize Disk Space 8-8
- Start All Required Virtual Processors 8-8
- Make Necessary Conversions 8-9
- Initiate Fast Recovery. 8-9
- Initiate a Checkpoint 8-9
- Document Configuration Changes 8-9
- Create the oncfg_servername.servernum File 8-10
- Drop Temporary Tblspaces. 8-10
- Set Forced Residency If Specified 8-10
- Return Control to User 8-11
- Prepare SMI Tables 8-11

After Initialization 8-12

Initialization of the database server refers to two related activities: disk-space initialization and shared-memory initialization. This chapter defines the two types of initialization and describes the activities that take place during initialization.

Types of Initialization

Shared-memory initialization establishes the contents of database server shared memory as follows: internal tables, buffers, and the shared-memory communication area.

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.

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.*

Two key differences distinguish shared-memory initialization from disk-space initialization:

- Shared-memory initialization has no effect on disk-space allocation or layout. No data is destroyed.
- Shared-memory initialization performs fast recovery.

Initialization Commands

You must be user **informix** or **root** to initialize the database server. The database server must be in off-line mode when you begin initialization. (For more information about modes, refer to Chapter 7, "Managing Database Server Operating Modes.")

You can initialize shared memory and disk space using one of the following methods:

- Informix Enterprise Command Center (IECC)

  For information on how to initialize shared-memory and disk space with IECC, refer to the *Informix Enterprise Command Center User Guide*.

- The **oninit** utility

  For more information, refer to "oninit: Initialize Dynamic Server" on page 35-20.

UNIX

- The ON-Monitor utility

  For more information, refer to "Using ON-Monitor" on page 32-3. ♦

The options that you include in the **oninit** command or, on UNIX, the options that you select from ON-Monitor determine the specific initialization procedure.

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.

The two lists in Figure 8-1 show the main tasks completed during the two types of initialization. Each step is discussed in the following sections.

Figure 8-1
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.
	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 on-line mode and return control to user.	Change to on-line mode and return control to user.
If the SMI tables are not current, update the tables.	Create SMI tables.

Process Configuration File

The database server uses configuration parameters to allocate shared-memory segments during initialization. If you change the size of shared memory by modifying a configuration-file parameter, you must take the database server to off-line mode and then reinitialize.

During initialization, the database server looks for configuration values in the following three files, in order:

1. If the **ONCONFIG** environment variable is set, the database server reads values from the following file.

Platform	ONCONFIG Variable
UNIX	**$INFORMIXDIR/etc/$ONCONFIG**
Windows NT	**%INFORMIXDIR%\etc\%ONCONFIG%**

   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.

Platform	ONCONFIG Variable
UNIX	**$INFORMIXDIR/etc/onconfig**
Windows NT	**%INFORMIXDIR%\etc\onconfig**

3. If the database server cannot find the **onconfig** file, it reads the configuration values from the following file.

Platform	ONCONFIG Variable
UNIX	**$INFORMIXDIR/etc/onconfig.std**
Windows NT	**%INFORMIXDIR%\etc\onconfig.std**

Informix recommends that you *always* set the **ONCONFIG** environment variable before you initialize the database server. The default configuration files are intended as templates and not as functional configurations. For more information about the configuration file, refer to step 2 on page 8-6.

The initialization 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. For more information about PAGE_CONFIG, refer to "PAGE_CONFIG" on page 38-8. Where differences exist, the database server uses the values from the current ONCONFIG configuration file for initialization.

Create Shared-Memory Portions

Next, 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.

The database server creates shared memory by acquiring the shared-memory space from the operating system for three different types of memory:

- Resident portion, used for data buffers, buffer tables, and so on
- Virtual portion, used for most internal 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. ♦

UNIX

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, refer to "The Virtual Portion of Shared Memory" on page 11-25.

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 Shared-Memory Structures

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. For example, 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, refer to "onstat: Monitor Database Server Operation" on page 35-62.

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. 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. For more information about PAGE_PZERO, refer to "Reserved Pages" on page 38-6.

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, refer to "What Is a Virtual Processor?" on page 9-5.

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, refer to the *Informix Migration Guide*.

Initiate Fast Recovery

The database server checks if fast recovery is needed and, if so, initiates it. For more information about fast recovery, refer to "What Is Fast Recovery?" on page 22-3.

Fast recovery is not performed during disk-space initialization because there is not yet anything to recover

Initiate a Checkpoint

After fast recovery executes, the database server initiates a checkpoint. As part of the checkpoint procedure, the database server writes a checkpoint-complete message in the message log. For more information about checkpoints, refer to "Checkpoints" on page 11-57.

The database server now moves to quiescent mode or on-line mode, depending on how you started the initialization 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. Where 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.

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 NT. The database server uses this file during a full-system restore. For more information about this file, refer to "oncfg_servername.servernum" on page A-8.

Drop Temporary Tblspaces

This task is not performed during disk-space initialization.

The database server searches through all dbspaces for temporary tblspaces. (If you initialize the database server using **oninit -p**, 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, refer to "What Is a Temporary Table?" on page 13-24.

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 parameter, refer to "RESIDENT" on page 33-74.

Return Control to User

After the previous steps are complete, the database server writes an `initialization complete` message in the message log. For more information about the message path, refer to "MSGPATH" on page 33-55.

At this point, control returns to the user. Any error messages generated by the initialization procedure are displayed in one or more of the following locations:

- The command line
- The database server message log file, specified by the MSGPATH environment variable
- Within ON-Monitor, if you are using it ♦
- Within IECC, if you are using it

UNIX

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. (For more information about SMI tables, refer to Chapter 34, "The sysmaster Database.") 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` into the message-log file.

If you shut down the database server before it finishes building the SMI tables, the process of building the tables aborts. 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 on-line. 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 or ON-Archive for dbspace or logical-log backups.

After Initialization

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. Informix recommends that you *do not* try to stop the database server by killing a virtual processor or another database server process. For more information, refer to "Starting and Stopping Virtual Processors" on page 10-7.

Disk, Memory, and Process Management

Section IV

Chapter

9

What Is Informix Dynamic Scalable Architecture?

What Is a Virtual Processor? 9-5
What Is a Thread? 9-5
What Is a User Thread? 9-7
Types of Virtual Processors 9-7
Advantages of Virtual Processors 9-9
Sharing Processing 9-9
Saving Memory and Resources 9-9
Processing in Parallel 9-10
Adding and Dropping Virtual Processors in On-Line Mode 9-11
Binding Virtual Processors to CPUs 9-12

How Virtual Processors Service Threads 9-12
Control Structures 9-13
Context Switching 9-14
Stacks 9-15
Queues 9-16
Ready Queues 9-17
Sleep Queues 9-17
Wait Queues 9-18
Multiple Virtual Processors on Windows NT 9-19
Mutexes 9-19

Virtual-Processor Classes 9-20
CPU Virtual Processors 9-20
How Many CPU Virtual Processors Do You Need? 9-20
Running on a Multiprocessor Computer 9-21
Running on a Single-Processor Computer 9-21
Adding and Dropping CPU Virtual Processors in On-Line Mode 9-22
Preventing Priority Aging 9-22
Using Processor Affinity 9-22

Disk I/O Virtual Processors 9-24
I/O Priorities 9-25
Logical-Log I/O 9-25
Physical-Log I/O 9-26
Asynchronous I/O 9-26
Network Virtual Processors. 9-28
Should Poll Threads Run on CPU or Network Virtual Processors?. 9-29
How Many Networking Virtual Processors Do You Need? 9-29
Listen and Poll Threads for the Client/Server Connection 9-30
Starting Multiple Listen Threads 9-33
Communications Support Module Virtual Processor 9-35
Optical Virtual Processor 9-35
Audit Virtual Processor 9-36
Miscellaneous Virtual Processor 9-36

Informix *Dynamic Scalable Architecture* (DSA) exploits the capabilities of both symmetric multiprocessor and uniprocessor platforms to deliver database scalability, manageability, and performance. This chapter discusses how the database server can scale its resources in relation to the demands that applications place on it. DSA provides the following performance and manageability advantages:

- A small number of database server processes can service a large number of client application processes, producing the following benefits:
  - Reduced operating-system overhead (fewer processes to run)
  - Reduced overall memory requirements
  - Reduced contention for resources within the DBMS
- DSA provides more control over setting priorities and scheduling database tasks than the operating system does.

Dynamic Server particularly exploits *symmetric multiprocessing* computer systems (SMPs). In a symmetric multiprocessing computer system, multiple CPUs (central processing units, or processors) all run a single copy of the operating system, sharing memory and communicating with each other as necessary. This chapter describes the following additional advantages that the database server provides on these systems:

- Multiple processes can work in parallel for one client.
- On some multiprocessor computers, you can bind database server processes to specific CPUs.

Figure 9-1 shows the three major components of the Informix database server architecture.

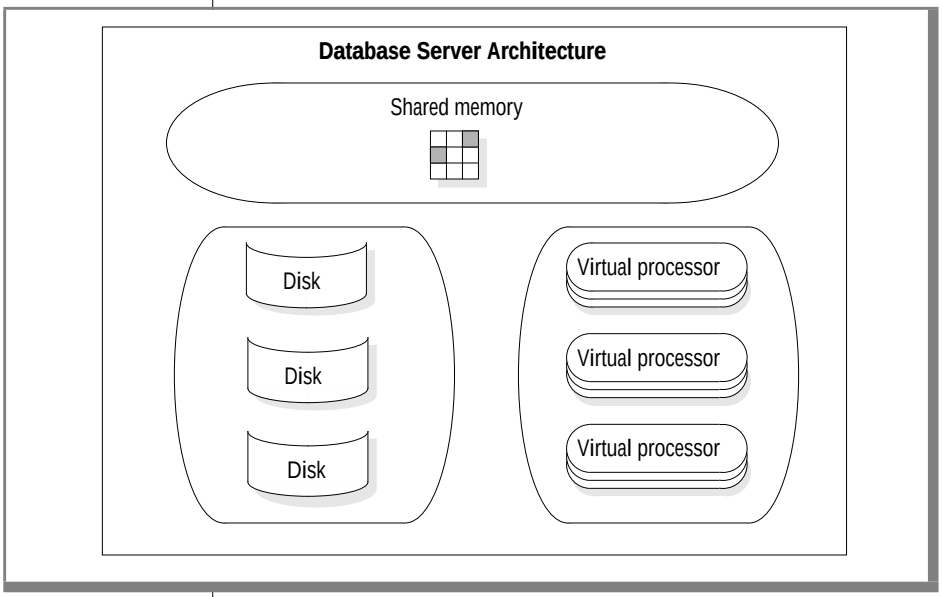


Figure 9-1
Database Server Architecture Components

The following are the three major components of the Informix database server architecture:

- Virtual-processor component

  A virtual processor is a task that the operating system schedules for execution on the CPU. Virtual processors and virtual-processor classes are the main topics of this chapter.

- Shared-memory component

  Shared memory consists of a resident and virtual portion. The database server uses shared memory to cache data from the disk in shared memory for faster access (this is resident shared memory). The database server also uses shared memory to maintain and control the resources required by the virtual processors (virtual). For more information on how the database server uses shared memory, see Chapter 11, "Shared Memory," and Chapter 12, "Managing Shared Memory."

- Disk Component

  The disk component is a collection of one or more units of disk space assigned to the database server system. All the data in the databases and all of the system information necessary to maintain the database server system reside within the disk component. For more information on the disk components, see Chapter 13, "Where Is Data Stored?"

What Is a Virtual Processor?

The central component of DSA is the *virtual processor*, which is a database server process. Database server processes are called virtual processors because they function similarly to the way that a CPU functions in a computer. Just as a CPU runs multiple operating-system processes to service multiple users, a virtual processor runs multiple *threads* to service multiple SQL client applications.

Virtual processors are *multithreaded processes* because they run multiple concurrent threads (MCT).

What Is a Thread?

A thread is a piece of work for a virtual processor in the same way that the virtual processor is a piece of work for the CPU. How the database server processes a thread depends on which operating system you are running. The following table provides descriptions of threads for each platform.

Platform	Description of Thread
UNIX	A thread is a task that the virtual processor schedules internally for processing.
Windows NT	A virtual processor is a task that the Windows NT operating system schedules. The database server runs within Windows NT threads, because each virtual processor is implemented as a Windows NT thread.

Important: *Throughout this chapter, all references to "thread" refer to the threads created, scheduled, and destroyed by Dynamic Server. All references to "NT threads" refer to the threads created, scheduled, and destroyed by the Windows NT operating system.*

Figure 9-2 illustrates the relationship of client applications to virtual processors.

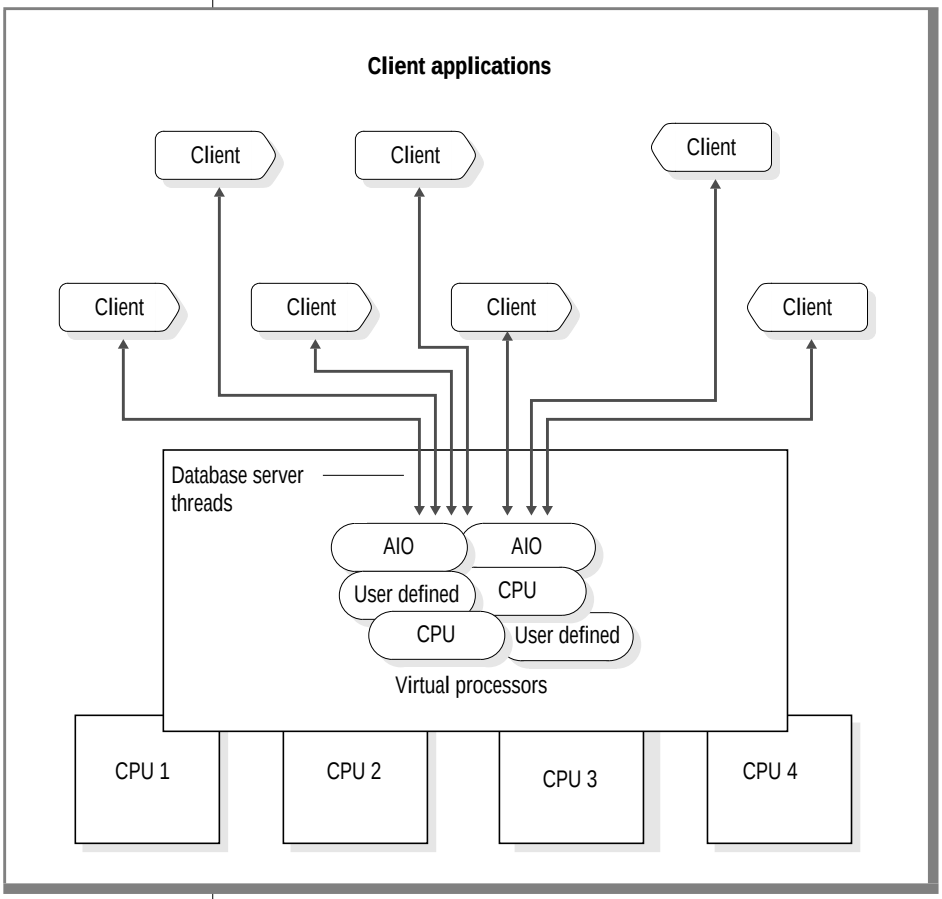


Figure 9-2
Dynamic Scalable Architecture and Virtual Processors

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 9-10.

What Is a User Thread?

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 ON-Monitor requests, a thread to service requests from the **onmode** utility, threads for recovery, and page-cleaner threads.

To display active user threads, use **onstat -u**, as explained in "Monitoring Sessions and Threads" on page 29-35.

Types of Virtual Processors

Virtual processors are divided into *classes* that are based on the type of processing that they do. Each class of virtual processor is dedicated to processing certain types of threads. Figure 9-3 on page 9-8 shows the classes of virtual processors and the types of processing that they do.

Figure 9-3
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 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 kernel asynchronous I/O is used, AIO virtual processors perform I/O to cooked disk spaces.
SHM	Network	Performs shared memory communication.
TLI	Network	Performs network communication using TLI.
SOC	Network	Performs network communication using sockets.
OPT	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.

UNIX

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 do 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 migrates the thread to another virtual processor because the benefit of balancing the processing load outweighs the amount of overhead incurred in transferring the process 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 a one-client-process-to-one-server-process architecture. 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 9-14.

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

Figure 9-4 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.

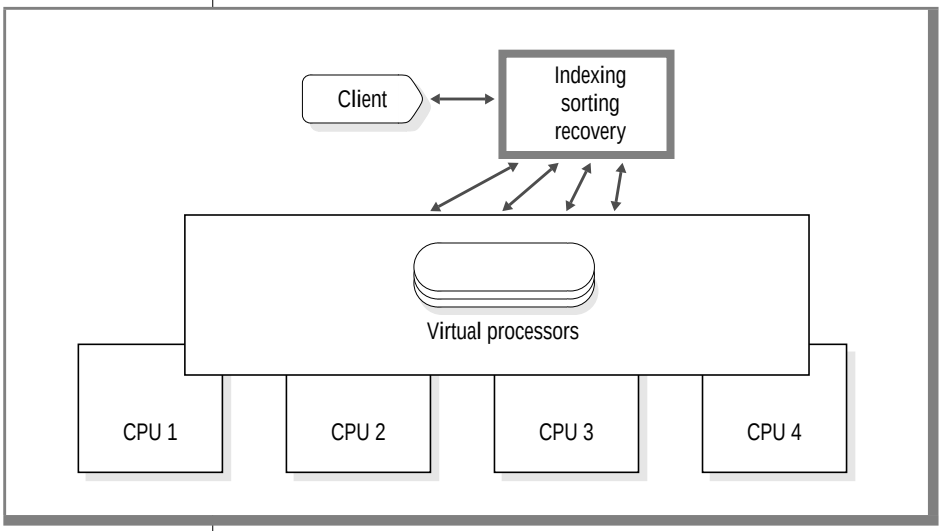


Figure 9-4
Parallel Processing

Adding and Dropping Virtual Processors in On-Line 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.

While the database server is running, you can add virtual processors for any of the classes, but you can only drop virtual processors for the CPU class. For information on how to add or drop virtual processors while the database server is in on-line mode, refer to "Adding Virtual Processors in On-Line Mode" on page 10-7 and "Dropping CPU Virtual Processors in On-Line Mode" on page 10-10.

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 virtual processors to specific CPUs in the computer. When you bind a 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 virtual processors to specific CPUs also enables you to isolate database work to 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 9-22.

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 complete 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 does 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

A thread yields at a predetermined point when the amount of processing required to complete a task would cause other threads to wait for an undue length of time. To alleviate this problem, the code for such tasks is written to include calls to the yield function at strategic points in the processing. When a thread performs one of these long-running tasks, it yields when it encounters one of these function calls. When a thread yields, other ready threads 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 9-5 on page 9-15 illustrates how a virtual processor accomplishes a context switch.

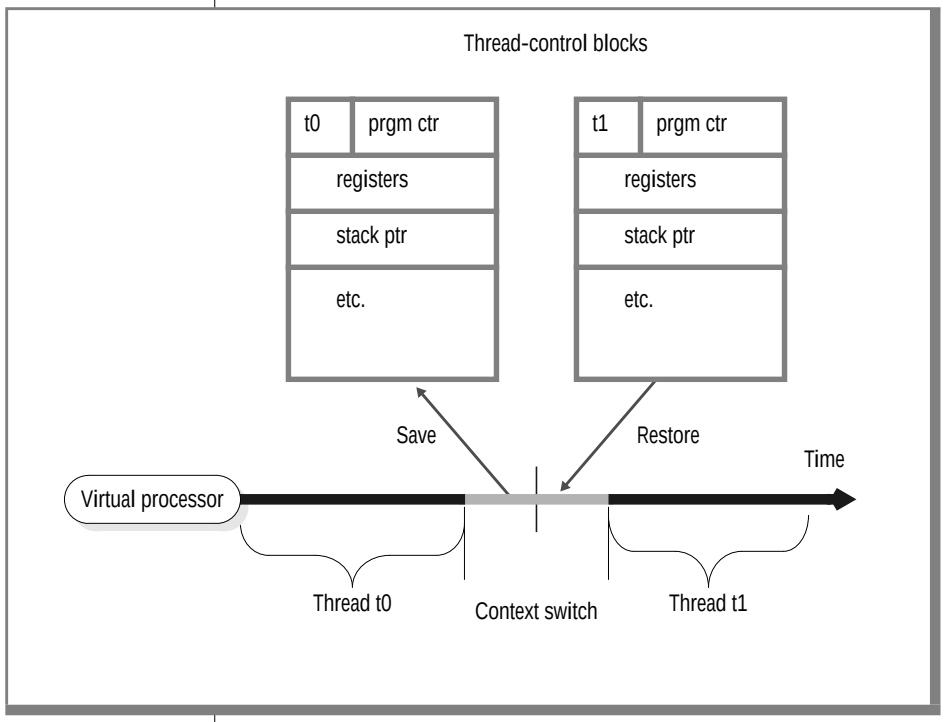


Figure 9-5 *Context Switch: How a Virtual Processor Switches from One Thread to Another*

Stacks

Dynamic Server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area of the thread is called the *stack*. For information on how to set the size of the stack, refer to "Stacks" on page 11-32.

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 9-6 illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.

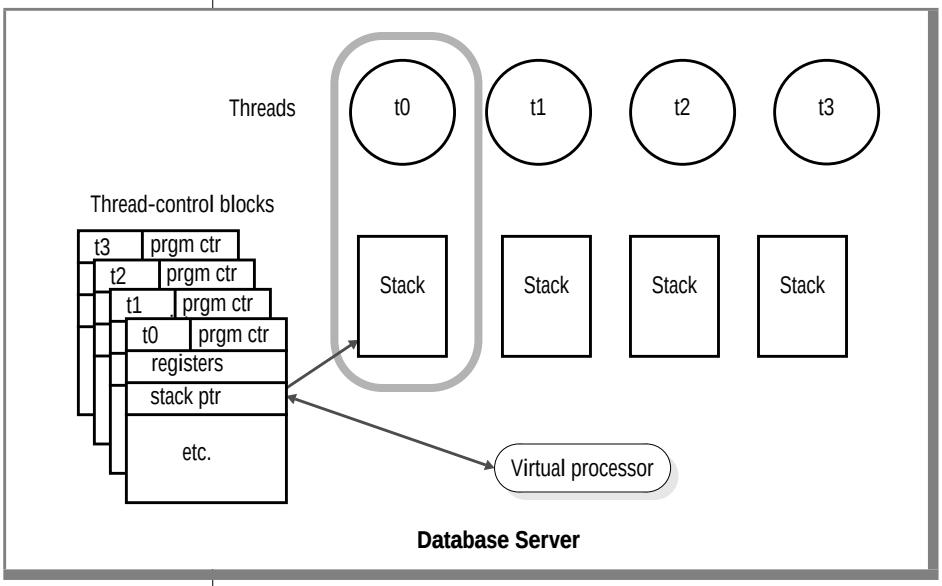


Figure 9-6
Virtual Processors Segregate Nonshared Data for Each User

Queues

Dynamic 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 29-33. For information on how to add virtual processors while the database server is in on-line mode, refer to "Adding Virtual Processors in On-Line Mode" on page 10-7.

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 is needed again—that is, 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 into a ready queue. Figure 9-7 illustrates how the database server threads are queued to perform database I/O.

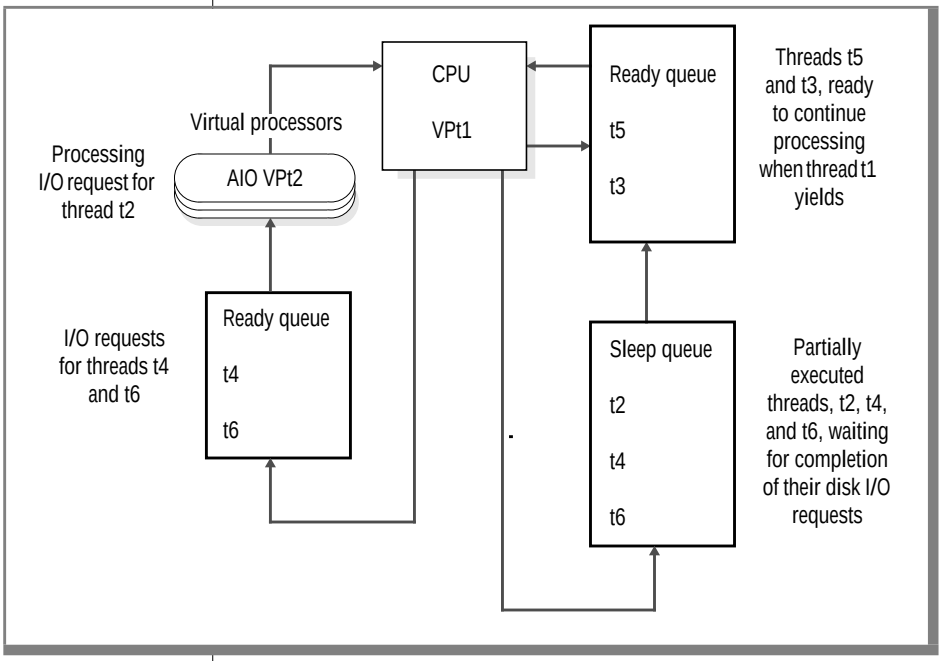


Figure 9-7
How Database Server Threads Are Queued to Perform Database I/O

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.

WIN NT

Multiple Virtual Processors on Windows NT

A database server running on Windows NT can take advantage of its fan-out capabilities. You can use multiple virtual processors (and multiple CPUs, if available) to work simultaneously, as Figure 9-8 illustrates.

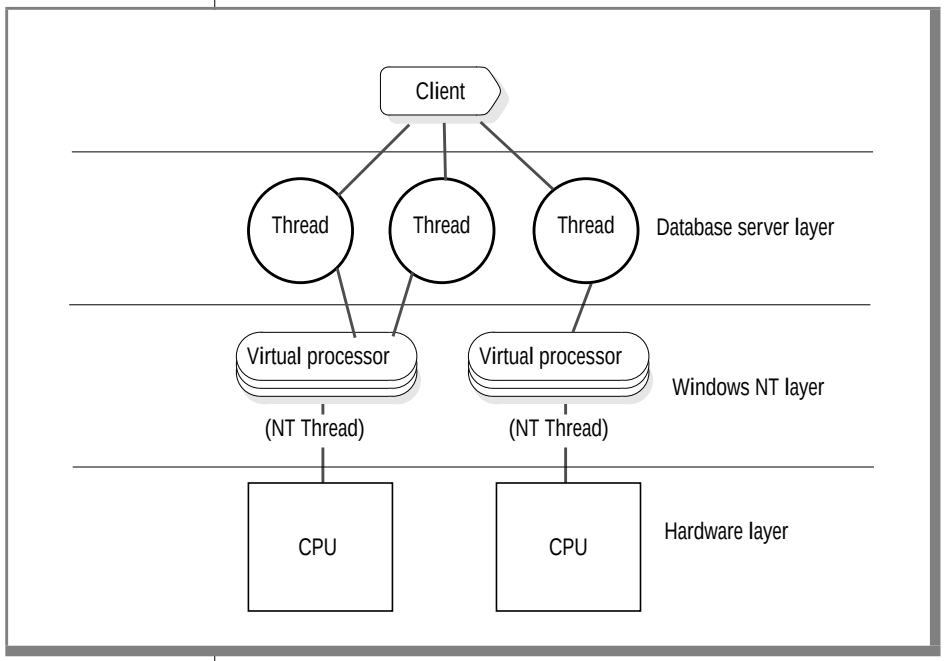


Figure 9-8
Fan-out capabilities on Windows NT

Important: *The database server creates multiple threads to do the work for one user for the following operations: sorting, indexing, and recovery.*

Mutexes

A mutex (*mut*ually *ex*clusive) is a latching mechanism that the database server uses to synchronize access by multiple threads to shared resources. Mutexes are similar to semaphores, which the operating system uses 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, once 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 (latches), refer to "Monitoring Latches" on page 29-27.

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 tells you 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.

How Many CPU Virtual Processors Do You Need?

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.

The NUMCPUVPS configuration parameter allows you to specify the number of CPU virtual processors that the database server starts initially. For information about the NUMCPUVPS parameter, refer to "NUMCPUVPS" on page 33-63.

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.

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 "MULTIPROCESSOR" on page 33-56.

Running on a Single-Processor Computer

If you are running only one CPU virtual processor, set the MULTIPROCESSOR configuration parameter to 0. 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 "MULTIPROCESSOR" on page 33-56.

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 "SINGLE_CPU_VP" on page 33-84.

Important: *You do not reduce the number of mutex calls by setting NUMCPUVPS to 1 and SINGLE_CPU_VP to 0, even though you are specifying only one CPU virtual processor.*

When you set the SINGLE_CPU_VP parameter to 1, the database server allows only one CPU virtual processor to run.

Adding and Dropping CPU Virtual Processors in On-Line Mode

You can add or drop CPU class virtual processors while the database server is on-line. For instructions on how to do this, see "Adding Virtual Processors in On-Line Mode" on page 10-7 and "Dropping CPU Virtual Processors in On-Line Mode" on page 10-10.

Preventing Priority Aging

Some operating systems decrement the priority of long-running processes as they accumulate processing time. This feature of the operating system is called *priority aging*. 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 described in "Documentation Notes, Release Notes, Machine Notes" on page 15 of the Introduction.

If your operating system allows you to disable priority aging, you can disable it by setting the NOAGE parameter. For more information on the NOAGE parameter, refer to "NOAGE" on page 33-61.

Using Processor Affinity

On some multiprocessor platforms that support *processor affinity*, you can assign virtual processors to specific CPUs. When you assign a virtual processor to a specific CPU, the virtual processor runs exclusively on that CPU. To see if processor affinity is supported on your database server platform, refer to the database server machine-notes file, which is described in "Documentation Notes, Release Notes, Machine Notes" on page 15 of the Introduction.

Figure 9-9 illustrates the concept of processor affinity. You can specify processor affinity by setting the AFF_SPROC and AFF_NPROCS parameters.

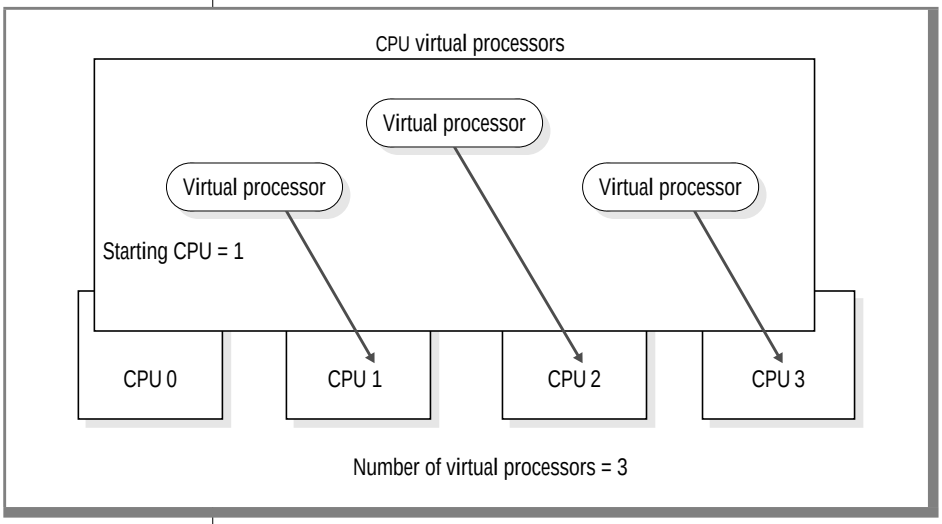


Figure 9-9
Processor Affinity

Setting Processor Affinity with the AFF_SPROC and AFF_NPROCS Parameters

You can also set the following two parameters in the ONCONFIG file to implement processor affinity on multiprocessor computers that support it:

- AFF_NPROCS
- AFF_SPROC

Set the AFF_NPROCS parameter to the number of CPUs to which you want to assign CPU virtual processors. The number of CPUs should be greater than the number of CPU virtual processors that you allocate. Also, do not set AFF_NPROCS to a number that is less than the number of virtual processors you have allocated.

Set the AFF_SPROC parameter to the number of the first CPU to which a CPU virtual processor should be assigned. The database server assigns CPU virtual processors to CPUs in serial fashion, starting with this processor. The first CPU is number 0. For example, if the computer has four CPUs and you set NUMCPUVPS to 3, AFF_SPROC to 1, and AFF_NPROCS to 3, the three CPU virtual processors are assigned to the second, third, and fourth CPUs, respectively.

The value of AFF_NPROCS plus the value of AFF_SPROC must be less than or equal to the number of CPUs. Using the previous example, if you set AFF_SPROC to 2, the database server would display an error message because 3 (AFF_NPROCS) plus 2 (AFF_SPROC) equals 5, and only 4 CPUs are available.

Disk I/O Virtual Processors

The following classes of virtual processors perform disk I/O:

- CPU
- AIO (asynchronous I/O)
- PIO (physical-log I/O)
- LIO (logical-log I/O)

The database server uses either the CPU class or the AIO class of virtual processors to perform all I/O that is not related to physical or logical logging. The database server uses the CPU class to perform *kernel-asynchronous I/O* (KAIO) when it is available on a platform. If the database server implements kernel-asynchronous I/O, a KAIO thread performs all I/O to raw disk space, including I/O to the physical and logical logs.

When kernel asynchronous I/O is not implemented, or when the I/O is to cooked disk space, the AIO class of virtual processors performs all nonlogging I/O. For more information about nonlogging I/O, refer to "Asynchronous I/O" on page 9-26.

Tip: *Review "Unbuffered or Buffered Disk Access" on page 13-6 to determine how your operating system supports raw and cooked disk space.*

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* they reside in raw disk space and the database server has implemented kernel asynchronous I/O.

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 Figure 9-10 shows.

Figure 9-10
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:

- Kernel asynchronous I/O is not implemented.
- The logical-log files are in cooked disk space.

Only when kernel asynchronous I/O 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:

- Kernel asynchronous I/O is not implemented.
- The physical-log file is in cooked disk space.

Only when kernel asynchronous I/O is implemented and the physical-log file is in raw disk space does the database server uses 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 11-57.) 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 either by requesting kernel asynchronous I/O, where available, through the CPU class of virtual processors or by using AIO virtual processors. 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 kernel-asynchronous I/O 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 kernel-asynchronous I/O 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.

Informix implements kernel-asynchronous I/O when it ports the database server to a platform that supports it. The database administrator does not implement kernel-asynchronous I/O. To see if kernel-asynchronous I/O is supported on your computer, refer to the database server machine-notes file, which is described under "Documentation Notes, Release Notes, Machine Notes" on page 15 of the Introduction.

AIO Virtual Processors

If the platform does not support kernel-asynchronous I/O, or if the I/O is to cooked disk space, 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 based on the filename of the chunk. Thus, each uniquely named chunk has its own queue, and chunks with the same name share a single queue. 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.

You use the NUMAIOVPS parameter to specify the number of AIO virtual processors that the database server starts initially. For information about the NUMAIOVPS parameter, refer to "NUMAIOVPS" on page 33-62.

You can start additional AIO virtual processors while the database server is in on-line mode. For more information, refer to "Adding Virtual Processors in On-Line Mode" on page 10-7.

You cannot drop AIO virtual processors while the database server is in on-line mode.

How Many AIO Virtual Processors Do You Need?

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 I/O request 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 allows you to monitor the length of the I/O queues for the AIO virtual processors. For more information, refer to "Monitoring Virtual Processors" on page 29-33.

If the database server implements kernel-asynchronous I/O on your platform, and all of your dbspaces are composed of raw disk space, one AIO virtual processor might be sufficient.

If the database server implements kernel-asynchronous I/O, but you are using some cooked file space, allocate two AIO virtual processors per active dbspace that is composed of cooked file space. If kernel-asynchronous I/O is *not* implemented on your platform, allocate two AIO virtual processors for each disk that the database server accesses frequently.

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.

Network Virtual Processors

As explained in Chapter 4, "Client/Server Communications," a client can connect to the database server in the following ways:

- Through a network connection
- Through a pipe
- Through shared memory

UNIX

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*).

Should Poll Threads Run on CPU or Network Virtual Processors?

Poll threads can run either *in-line* on CPU virtual processors or, depending on the connection type, 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 `vp 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 on-line mode, you cannot drop a CPU virtual processor that is running a poll thread.

How Many Networking Virtual Processors Do You Need?

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.

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. You specify a listen port for each of these dbservername entries by assigning it a unique combination of **hostname** and **service name** entries in **sqlhosts**. For example, the **sqlhosts** file or registry entry shown in Figure 9-11 causes the database server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

Figure 9-11
A Listen Thread for Each Listen Port

dbservername	nettype	hostname	service name	options
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 "How Many Networking Virtual Processors Do You Need?" on page 9-29.

For information on how to specify whether the poll threads for an interface / protocol combination run in CPU or network virtual processors, refer to "Should Poll Threads Run on CPU or Network Virtual Processors?" on page 9-29 and "NETTYPE" on page 33-57.

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 does the primary processing for the client. Figure 9-12 illustrates the roles of the listen and poll threads in establishing a connection with a client application.

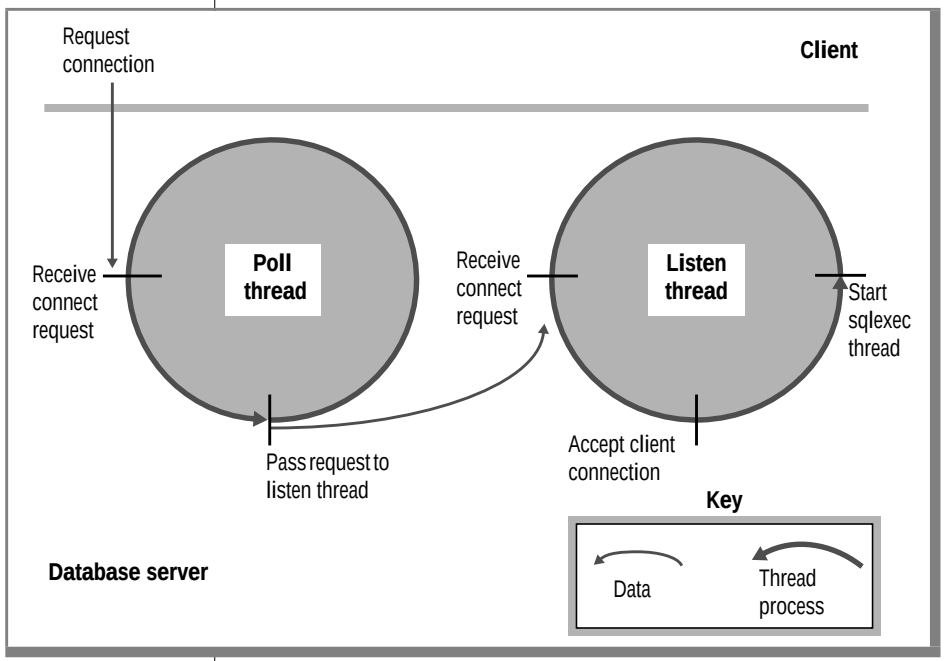


Figure 9-12
The Roles of the Poll and the Listen Threads in Connecting to a Client

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

For a shared-memory connection, the poll thread places the message in the communications portion of shared memory. ♦

Figure 9-13 illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

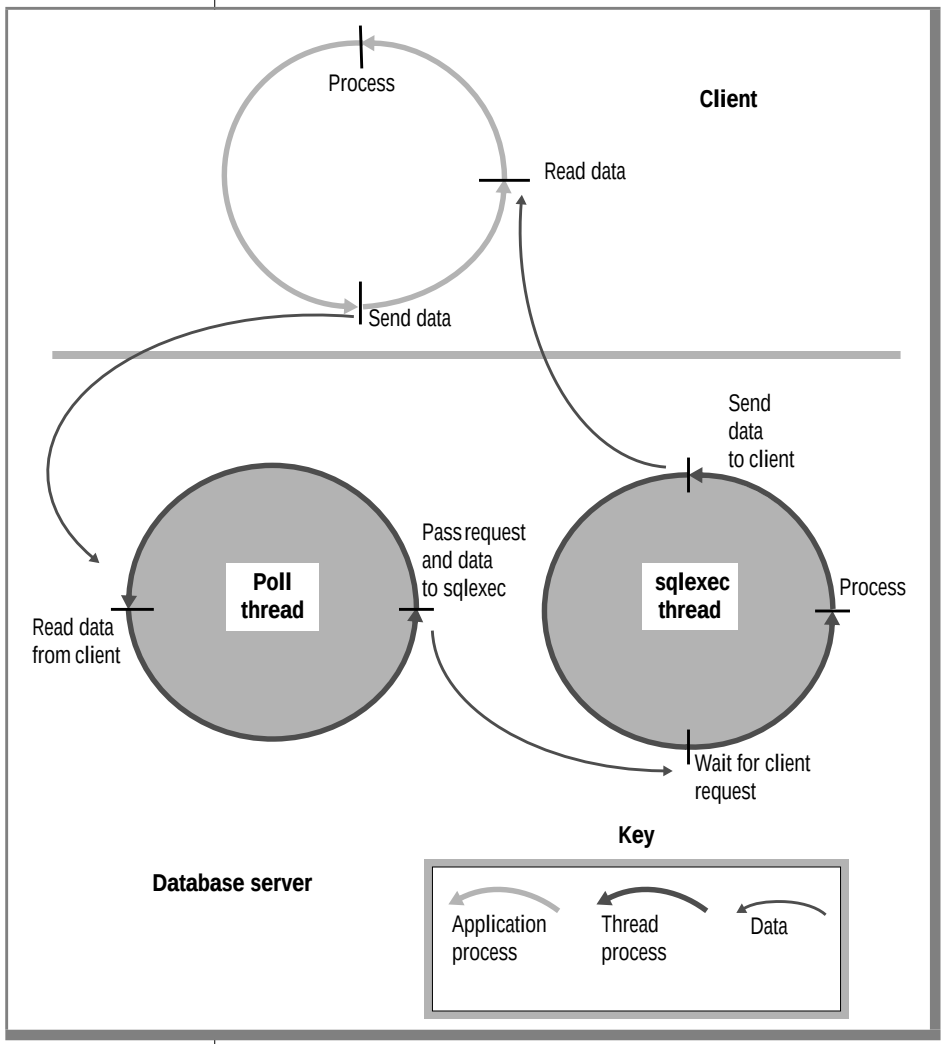


Figure 9-13
The Roles of the Poll and sqlexec Threads in Communicating with the Client Application

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 specify dbservernames for each of the ports using the DBSERVERALIASES parameter. For example, the DBSERVERALIASES parameter in Figure 9-14 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 9-14
Defining Multiple dbservernames for Multiple Connections of the Same Type

Once 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 Figure 9-15 on page 9-34 cause the database server to start three listen threads for the **onsoctcp** interface / protocol combination, one for each of the ports defined.

Figure 9-15
Sqlhosts Entries to Listen to Multiple Ports for a Single Interface/Protocol Combination

dbservername	nettype	hostname	service name	options
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 Figure 9-15, 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 "The sqlhosts File or Registry" on page 4-27.

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 9-14, the **sqlhosts** file or registry entries shown in Figure 9-16 on page 9-35 cause the database server to start three listen threads for the same interface / protocol combination (as did the entries in Figure 9-15). 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**).

Figure 9-16
Example of sqlhosts Entries to Support Two Network-Interface Cards for the onsoctcp Interface/Protocol Combination

dbservername	nettype	hostname	service name	options
soc_ol1	onsoctcp	myhost1	port1	
soc_ol2	onsoctcp	myhost1	port2	
soc_ol3	onsoctcp	myhost2	port1	

UNIX

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 executes a number of CSM virtual processors equal to the number of CPU virtual processors defined in the NUMCPUVPS configuration parameter.

For more information on the Communications Support Service, refer to Chapter 4, "Client / Server Communications."

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 Optical Subsystem, see the *Guide to the Optical Subsystem*.

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, refer to your *Trusted Facility Manual*.

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

10

Managing Virtual Processors

Setting Virtual-Processor Configuration Parameters 10-3
Setting Virtual-Processor Configuration Parameters with a Text Editor 10-4
Setting Virtual-Processor Configuration Parameters with ON-Monitor 10-5

Starting and Stopping Virtual Processors 10-7
Adding Virtual Processors in On-Line Mode. 10-7
Using onmode to Add Virtual Processors in On-Line Mode 10-8
Using ON-Monitor to Add Virtual Processors in On-Line Mode 10-8
Adding Network Virtual Processors 10-10
Dropping CPU Virtual Processors in On-Line Mode 10-10

This chapter describes how to set the configuration parameters that affect database server virtual processors. This chapter also tells you 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 9, "What Is Informix Dynamic Scalable Architecture?"

Setting Virtual-Processor Configuration Parameters

As **root** or user **informix**, you can set the configuration parameters for the database server virtual processors with the following tools:

UNIX

- A text editor
- ON-Monitor

ON-Monitor does not allow you to change all configuration parameters. This chapter provides descriptions of configuration parameters, including the ON-Monitor menu sequences that lead you to the screen entries for each parameter. ♦

To put changes to parameters into effect, you must reinitialize shared memory. For information on how to reinitialize shared memory, refer to "Reinitializing Shared Memory" on page 12-14.

For more information on configuration parameters refer to Chapter 33, "Configuration Parameters."

Setting Virtual-Processor Configuration Parameters with a Text Editor

You can use a text editor program to set ONCONFIG parameters at any time. To change one of the virtual-processor configuration parameters, use the editor to locate the parameter in the file, enter the new value(s), and rewrite the file to disk.

Figure 10-1 lists the ONCONFIG parameters that are used to configure virtual processors. The page references in the third column refer to descriptions of the parameters in Chapter 33, "Configuration Parameters."

Figure 10-1
ONCONFIG Parameters for Configuring Virtual Processors

Parameter	Purpose	Reference
AFF_NPROCS	Specifies the number of CPUs to which CPU virtual processors will be assigned (multiprocessor computers only)	page 33-9
AFF_SPROC	Specifies the first CPU (of AFF_NPROCS) to which a CPU virtual processor will be assigned	page 33-10
MULTIPROCESSOR	Specifies that you are running on a multiprocessor computer	page 33-56
NETTYPE	Specifies parameters for network protocol threads (and virtual processors)	page 33-57
NOAGE	Specifies no priority aging of processes by the operating system	page 33-61
NUMAIOVPS	Specifies the number of AIO virtual processors	page 33-62
NUMCPUVPS	Specifies the number of CPU virtual processors	page 33-63
SINGLE_CPU_VP	Specifies that you are running a single CPU virtual processor	page 33-84

For more information on how these parameters affect virtual processors refer to "Virtual-Processor Classes" on page 9-20.

UNIX

Setting Virtual-Processor Configuration Parameters with ON-Monitor

To set the virtual-processor configuration parameters with ON-Monitor, select Parameters from the main menu, and then select the perFormance option.

```
   PERFORMANCE: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                   PERFORMANCE TUNING PARAMETERS

Multiprocessor Machine    [MULTIPROCESSOR]
  Num Procs to Affinity   [AFF_NPROCS]
  Proc num to start with  [AFF_SPROC]

CPU VPs             [NUMCPUVPS]
AIO VPs             [NUMAIOVPS]
Single CPU VP   [SINGLE_CPU_VP]     NETTYPE settings:
                                      Protocol Threads Users VP-class
Disable Priority Aging  [NOAGE]       [ipcshm]      [NETTYPE]
                                      [soctcp]      [NETTYPE]
```

Figure 10-2
Partial View of ON-Monitor PerFormance Screen with the ONCONFIG Parameter for the Virtual-Processor Entries

Figure 10-3 shows the full perFormance screen; the shaded entries set configuration parameters for virtual processors.

Figure 10-3
ON-Monitor PerFormance Screen

```
   PERFORMANCE: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-
level help.
                    PERFORMANCE TUNING PARAMETERS

Multiprocessor Machine       [N]      LRU Max Dirty             [ 60]
  Num Procs to Affinity   [  0]      LRU Min Dirty             [ 50]
  Proc num to start with  [  0]      Checkpoint Interval      [ 300]
                                     Num of Read Ahead Pages   [ 50]
CPU VPs                   [  5]      Read Ahead Threshold      [ 20]
AIO VPs                   [  1]
Single CPU VP               [N]    NETTYPE settings:
Use OS Time                 [N]      Protocol Threads Users VP-class
Disable Priority Aging      [N]      [ipcshm] [  2]   [  5] [CPU]
Off-Line Recovery Threads [ 10]      [soctcp] [  2]   [  5] [    ]
On-Line Recovery Threads  [  1]
Num of LRUS queues        [  8]


Are you running on a multiprocessor machine?
```

Figure 10-2 on page 10-5 shows only the perFormance screen entries for configuring virtual processors. For each entry, it shows the name of the associated parameter in the ONCONFIG file within a pair of brackets ([ ]).

Each row of entries under `NETTYPE settings` describes a separate NETTYPE parameter, one for each of the protocols available on the computer. The four columns for these entries (`Protocol`, `Threads`, `Users`, and `VP-class`) correspond to the four fields of the NETTYPE parameter.

For more information on the ONCONFIG parameters that are associated with the database server virtual processors, refer to Figure 10-1 on page 10-4.

Starting and Stopping Virtual Processors

When you start the **oninit** process to start the database server, **oninit** 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 9-20.

Once the database server is in on-line mode, you can start additional virtual processors to improve performance, if necessary. The database server allows you to start a maximum of 1,000 virtual processors. For more information, refer to "Adding Virtual Processors in On-Line Mode".

While the database server is in on-line mode, you can drop only virtual processors of the CPU class. For more information, refer to "Dropping CPU Virtual Processors in On-Line Mode" on page 10-10.

To terminate the database server and thereby terminate all virtual processors, use the **-k** option of the **onmode** utility. For more information on using the **-k** option of the **onmode** utility, refer to "Change Database Server Modes" on page 35-30.

Adding Virtual Processors in On-Line Mode

While the database server is in on-line mode, you can start additional virtual processors for the following classes: CPU, AIO, PIO, LIO, SHM, STR, TLI, SOC. You can start additional virtual processors for the CPU, AIO, PIO, LIO, SHM, STR, TLI, and SOC classes in one of the following two ways:

- Using the **-p** option of the **onmode** utility
- Using ON-Monitor ♦

UNIX

For the format of the **onmode** command, refer to "onmode: Mode and Shared-Memory Changes" on page 35-28.

Using onmode to Add Virtual Processors in On-Line Mode

Use the **-p** option of the **onmode** command to add virtual processors while the database server is in on-line mode. Specify the number of virtual processors that you want to add with a positive number that is greater than the number of virtual processors that are currently running. 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.

UNIX

Using ON-Monitor to Add Virtual Processors in On-Line Mode

To use ON-Monitor to add virtual processors while the database server is in on-line mode, select Modes from the main menu, and then select Add-Proc.

Figure 10-4 on page 10-9 shows the ON-Monitor Add-Proc screen, which allows you to add virtual processors in the following classes: CPU, AIO, LIO, PIO, and network.

```
DD VIRTUAL PROCESSORS: Press ESC to add virtual processors.
 Press Interrupt to cancel and return to the modes menu.


                        ADDING VIRTUAL PROCESSORS

 Number of CPU Virtual Processors to add                    [  0]

 Number of Asynchronous IO Virtual Processors to add        [  0]

 Number of Logical log IO Virtual Processors to add         [  0]

 Number of Physical log IO Virtual Processors to add        [  0]

 Number of Network Virtual Processors to add                [   ] [      ]
                                                            [   ] [      ]
                                                            [   ] [      ]




Enter the number of asynchronous IO processors to add
```

Figure 10-4
The ON-Monitor Add-Proc Screen

The logical-log and physical-log entries on the Add-Proc screen allow you to enter a number greater than 2, but the database server will not start more than two virtual processors in either of these classes. The database server automatically starts one virtual processor in each of these classes unless mirroring is used, in which case it starts two.

You specify network virtual processors by first entering the number of virtual processors and then entering 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-designated class. For information on adding virtual processors of a user-designated class in on-line mode, refer to "Using onmode to Add Virtual Processors in On-Line Mode" on page 10-8.

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 on-line mode, and you have specified in the NETTYPE 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.

Dropping CPU Virtual Processors in On-Line Mode

While the database server is in on-line mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, CPU 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 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 on CPU virtual processors and poll threads, refer to "Should Poll Threads Run on CPU or Network Virtual Processors?" on page 9-29.

Chapter

11

Shared Memory

What Is Shared Memory?. 11-5

How the Database Server Uses Shared Memory 11-6
How the Database Server Allocates Shared Memory 11-7
How Much Shared Memory Does the Database Server Need?. . . 11-9
What Action Should You Take If SHMTOTAL Is Exceeded? . . . 11-10

What Processes Attach to Shared Memory? 11-11
How a Client Attaches to the Communications Portion 11-11
Where the Client Attaches to the Communications Portion 11-11
How Utilities Attach to Shared Memory 11-12
How Virtual Processors Attach to Shared Memory 11-12
Defining a Unique Key Value 11-13
Specifying Where to Attach the First Shared-Memory Segment 11-14
How Virtual Processors Attach Additional Shared-Memory Segments 11-15
The Shared-Memory Lower-Boundary Address 11-15

Keeping Shared-Memory Segments Resident 11-17

The Resident Portion of Shared Memory 11-18
Shared-Memory Header. 11-19
Shared-Memory Buffer Pool 11-19
Regular Buffers 11-19
Logical-Log Buffer 11-22
Physical-Log Buffer 11-24
High-Availability Data-Replication Buffer 11-24

The Virtual Portion of Shared Memory 11-25
How the Database Server Manages the Virtual Portion of Shared Memory 11-25
How to Specify the Size of the Virtual Portion of Shared Memory. 11-25
The Virtual Portion of Shared Memory 11-26
Shared-Memory Internal Tables 11-26
Big Buffers 11-31
Session Data 11-31
Thread Data. 11-32
Dictionary Cache 11-33
Sorting Memory 11-33
Stored Procedures Cache 11-34
Global Pool 11-34

The Communications Portion of Shared Memory 11-34

Concurrency Control 11-35
Shared-Memory Mutexes 11-35
Shared-Memory Buffer Locks 11-36
Types of Buffer Locks 11-36

How Database Server Threads Access Shared Buffers 11-37
LRU Queues 11-37
LRU Queue Components 11-37
Why Are Pages Ordered in Least-Recently Used Order? . . . 11-38
LRU Queues and Buffer-Pool Management 11-38
How Many LRU Queues Should You Configure? 11-39
How Many Cleaners Should You Allocate? 11-40
Limiting the Number of Pages Added to the MLRU Queues 11-40
When MLRU Cleaning Ends 11-41
Configuring the Database Server to Read Ahead 11-42
How a Database Server Thread Accesses a Buffer Page 11-43
Identify the Page 11-43
Determine the Level of Lock Access 11-43
Try to Locate the Page in Shared Memory 11-44
Locate a Buffer and Read Page from Disk 11-44
Lock the Buffer If Necessary 11-44
Release the Buffer Lock and Wake a Waiting Thread 11-45

How the Database Server Flushes Data to Disk 11-46
Events That Prompt Flushing of the Regular Buffers 11-47
Flushing Before-Images First 11-47
Flushing the Physical-Log Buffer 11-47
Events That Prompt Flushing of the Physical-Log Buffer . . . 11-48
When the Physical-Log Buffer Becomes Full 11-49
How the Database Server Synchronizes Buffer Flushing 11-50
Ensuring That Physical-Log Buffers Are Flushed First. 11-50
How Write Types Describe Flushing Activity 11-51
Foreground Write 11-52
LRU Write 11-52
Chunk Write 11-53
Flushing the Logical-Log Buffer. 11-53
When the Logical-Log Buffer Becomes Full 11-54
After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging 11-55
When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates 11-55
When a Checkpoint Occurs 11-55
When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File. 11-56

How the Database Server Achieves Data Consistency 11-56
Critical Sections 11-56
Checkpoints 11-57
Events That Initiate a Checkpoint 11-57
Checkpoint Is Critical to Fast Recovery 11-59
Time Stamps 11-60
Time Stamps on Disk Pages 11-60
Time Stamps on Blobpages 11-61
Blob Time Stamps with Dirty Read and Committed Read Isolation Levels 11-61

Buffering TEXT and BYTE Data Types 11-63
Writing TEXT and BYTE Data 11-63
Blobpages Do Not Pass Through Shared Memory 11-63
TEXT and BYTE Objects Are Created Before the Data Row Is Inserted 11-64
Blobpage Buffers Are Created for the Duration of the Write . . . 11-64

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 12, "Managing Shared Memory."

What Is 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 uses 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.

How the Database Server Uses Shared Memory

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 11-1 illustrates the shared-memory scheme.

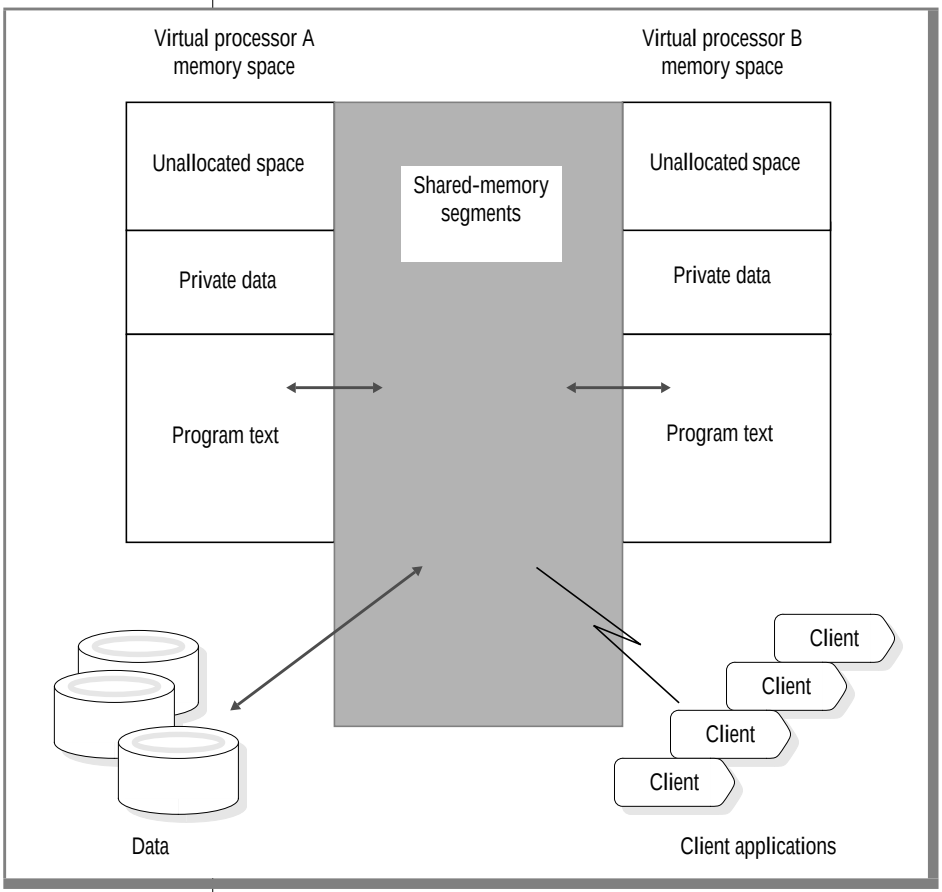


Figure 11-1 *How the Database Server Uses Shared Memory*

How the Database Server Allocates Shared Memory

The database server creates the following portions of shared memory:

- The *resident* portion
- The *virtual* portion
- The *IPC communications* or message portion ♦

UNIX

Each portion of shared memory consumes one or more operating-system segments. When the database server initializes shared memory, it allocates at least two operating-system segments, one for the resident portion and one for the virtual portion. It might allocate more segments if the maximum segment size is not large enough.

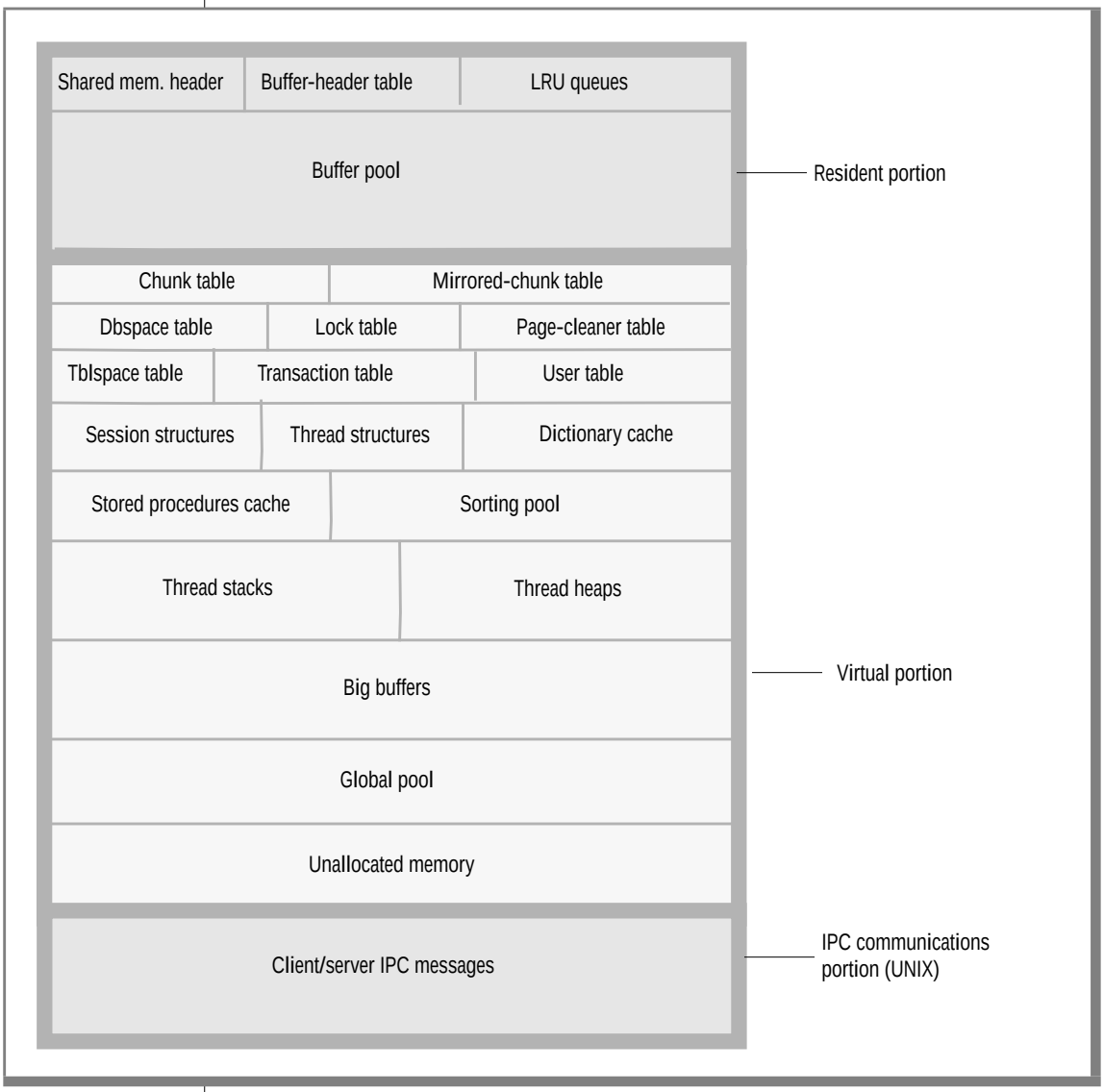
UNIX

If **sqlhosts** specifies shared-memory communications, the database server allocates memory for the communications portion. ♦

The database server adds operating-system segments as needed to the virtual portions of shared memory. Figure 11-2 on page 11-8 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 off-line mode to quiescent mode, or from off-line mode directly to on-line mode. The database server uses locks and latches to manage concurrent access to shared-memory resources by multiple threads. For more information about modes, refer to Chapter 7, “Managing Database Server Operating Modes.”

Figure 11-2
Contents of Dynamic Server Shared Memory



How Much Shared Memory Does the Database Server Need?

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 8 kilobytes in size and managed by a bit map.

The header-line output by the **onstat** utility contains the size of the database server shared memory, expressed in kilobytes. For information on how to use the **onstat** utility, refer to "onstat: Monitor Database Server Operation" on page 35-62. You can also use the **-g seg** option of **onstat** to monitor how much memory the database server allocates for each of the resident, virtual, and communications portions of shared memory.

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. 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: Dynamic Server may be unable to complete a checkpoint
17:19:13   Action: Make more virtual memory available to Dynamic 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 roll back, operations will no longer fail from lack of memory, and the database server continues to process transactions as usual.

What Action Should You Take If SHMTOTAL Is Exceeded?

The condition that occurs when the database server needs more memory than is allowed by SHMTOTAL is a transient condition, 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 you see messages on a regular basis that indicate that the database server needs more memory than is allowed by SHMTOTAL, you have not configured the database server correctly. Lowering the value of BUFFERS or DS_TOTAL_MEMORY is one possible solution, and increasing the value of SHMTOTAL is another.

What Processes Attach to Shared Memory?

The following processes attach to the database server shared memory:

UNIX

- 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.

UNIX

How a Client Attaches to the Communications Portion

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 4, "Client/Server Communications" and "Network Virtual Processors" on page 9-28.

Where the Client Attaches to the Communications Portion

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. By specifying 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 *Informix Guide to SQL: Reference*.

How Utilities Attach to Shared Memory

The database server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory through one of the files listed in the following table.

Platform	File
UNIX	**$INFORMIXDIR/etc/.infos.*servername***
Windows NT	**%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 initialization. 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:

1. Accesses the SERVERNUM parameter from the ONCONFIG file
2. Uses SERVERNUM TO calculate a shared-memory key value
3. 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.
4. Directs the operating system to attach the first shared-memory segment to its process space at SHMBASE
5. 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.

Defining a Unique Key Value

The database server uses the ONCONFIG parameter SERVERNUM to calculate a unique key value for its shared-memory segments. All virtual processors within a single database server instance share the same key value. When each virtual processor attaches to shared memory, it calculates the key value as follows:

```
(SERVERNUM * 65536) + shmkey
```

The value of *shmkey* is set internally and cannot be changed by the user. (The *shmkey* value is 52564801 in hexadecimal representation or 1,381,386,241 in decimal.) The value (SERVERNUM * 65,536) is the same as multiplying SERVERNUM by hexadecimal 10,000.

When more than one database server instance exists on a single computer, the difference in the key values for any two instances is the difference between the two SERVERNUM values, multiplied by 65,536.

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: *Informix recommends that you do not attempt to change the value of SHMBASE for the following reasons:*

- *The specific value of SHMBASE is often computer dependent. It is not an arbitrary number. Informix selects a value for SHMBASE that keeps 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.*

UNIX

- *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.* ♦

How Virtual Processors Attach 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 into 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.

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 11-3 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. For a description of the operating-system kernel parameter, refer to "Shared-Memory Lower-Boundary Address" on page 12-5.

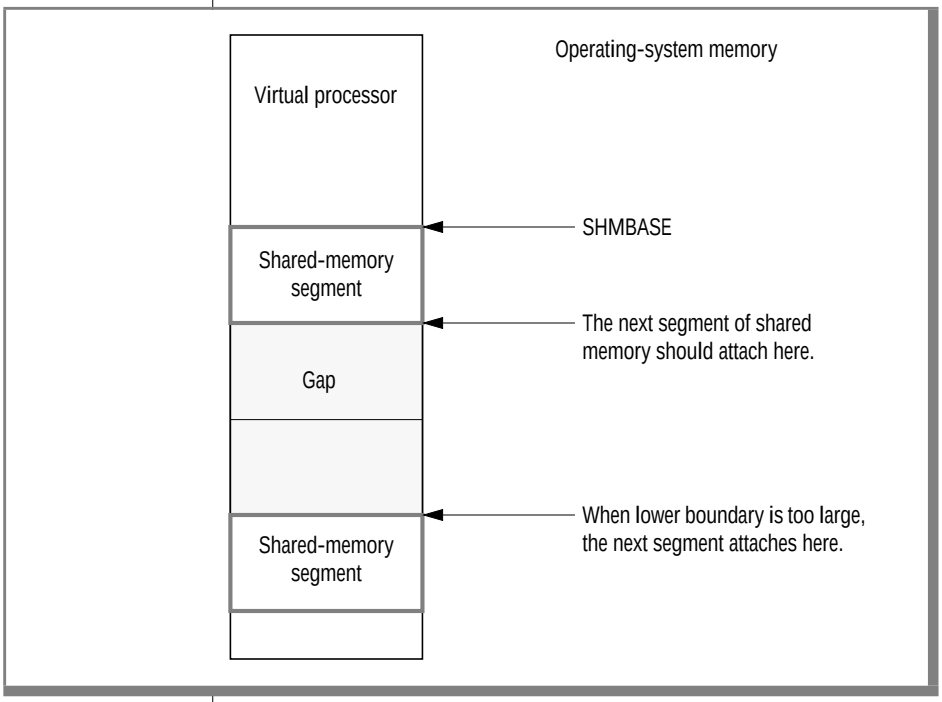


Figure 11-3
Shared-Memory Lower-Boundary Address Overview

Keeping Shared-Memory Segments Resident

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 resident and 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 (zero).

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 parameter, refer to "RESIDENT" on page 33-74.

The 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:

- Shared-memory header
- Buffer pool

Figure 11-4 illustrates the contents of the resident portion of shared memory.

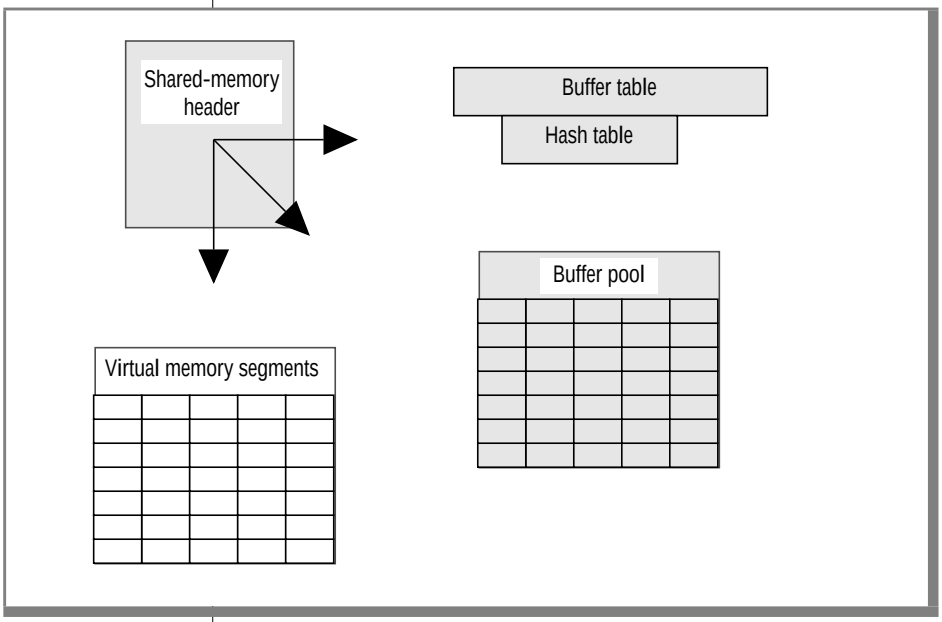


Figure 11-4
The Resident Portion of Shared Memory (Shaded Areas)

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 one kilobyte, 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 regular buffers that store database data pages.

If data pages are modified, entries are usually made in the following two other shared-memory buffers, also in the resident portion of shared memory, which function solely to ensure the physical and logical consistency of the data:

- Logical-log buffer
- Physical-log buffer

Regular Buffers

The regular buffers store dbspace pages read from disk. The pool of regular buffers comprises the largest allocation of the resident portion of shared memory.

You specify the number of regular buffers in the buffer pool with the BUFFERS parameter in the ONCONFIG file. BUFFERS defaults to 1000 buffers. Informix recommends that you allocate at least four buffers per user thread up to 2000 buffers. For more than 500 users, the minimum requirement is 2000 buffers. The following table lists the maximum number of allocated buffers for each platform.

Platform	Maximum Number of Allocated Buffers
UNIX	768 kilobytes (768 * 1024)
Windows NT	512 kilobytes

For more information on setting the BUFFERS configuration parameter, refer to "BUFFERS" on page 33-12.

The status of the regular buffers is tracked through the buffer table. Within shared memory, regular buffers are organized into LRU buffer queues. Buffer acquisition is managed through the use of latches, called *mutexes*, and lock-access information. You can monitor buffers and buffer-pool activity using four options of **onstat**:

- The **-b** and **-B** options display general buffer information.
- The **-R** option displays LRU queue statistics.
- The **-X** option displays information about the database server I/O threads that are waiting for buffers.

For a description of how LRU queues work, refer to "LRU Queues" on page 11-37. For a description of mutexes, refer to "Mutexes" on page 9-19.

How Big Is a Regular Buffer?

Each regular 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 regular buffer. The two exceptions are I/O performed from big buffers (see "Big Buffers" on page 11-31) and I/O performed from blobspace buffers (see "Blobpage Buffers Are Created for the Duration of the Write" on page 11-64). The **-b** option of the **onstat** utility also displays the database server page size. For information on the **onstat** utility, refer to "onstat: Monitor Database Server Operation" on page 35-62.

UNIX

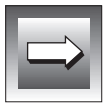
On UNIX. you also can determine the page size for your system by choosing **Parameters →Shared-memory** in ON-Monitor. ON-Monitor displays a list of shared-memory parameters, of which the database server page size is the last entry on the page. ♦

Memory-Resident Tables

You can specify that one or more fragments of a table or index remain resident in shared memory as much as possible. Any pages that belong to a resident fragment, either currently in the buffer or loaded in the future, are considered last for replacement when a free buffer is requested.

Use the SET TABLE and SET INDEX statements to turn on the resident state for one or more fragments of a table or index. For information on the SET TABLE and SET INDEX statements, refer to the *Informix Guide to SQL: Syntax*.

You must be the a database system administrator (DBSA) or user **informix** to turn on or change the resident state. As the DBSA, you can create a DBA-privileged stored procedure to grant the privilege to other users.



Important: *The resident state of a table fragment is not persistent; it must be specified each time that you start the server. The default state is nonresident.*

If you add new fragments to a resident table, the new fragments are not automatically marked as resident. You must use SET TABLE or SET INDEX to declare the new fragments as resident or redeclare the table as resident. If you use the SET TABLE or SET INDEX statement to turn residency off, or if you use the ALTER FRAGMENT statement to drop a fragment, its residency is cleared.

When you turn on residency, the buffer pool is scanned and any buffers that belong to these fragments are marked as the highest priority. When you turn off residency, the buffer pool is scanned and all buffers belonging to that partition are put back into the buffer pool with the default priority. When the fragment is dropped, the resident state of all the buffers for that partition in the buffer pool is cleared and put back into the buffer pool with the lowest priority.

The DBSA must keep track of how many tables are marked resident. If the buffer pool contains too many resident pages, the caching effect is defeated. The number of resident pages that can be held in memory is exceeded and high-priority pages are swapped to disk and replaced with other high-priority pages.

You can use the **-p** and **-P** options of the **onstat** utility to determine the rate of buffer cache usage and to optimize the use of memory resident tables, respectively. The **onstat -t** option enables you to determine which tblspaces are resident. For information on the **onstat** utility, refer to "onstat: Monitor Database Server Operation" on page 35-62.

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 11-5 on page 11-23.

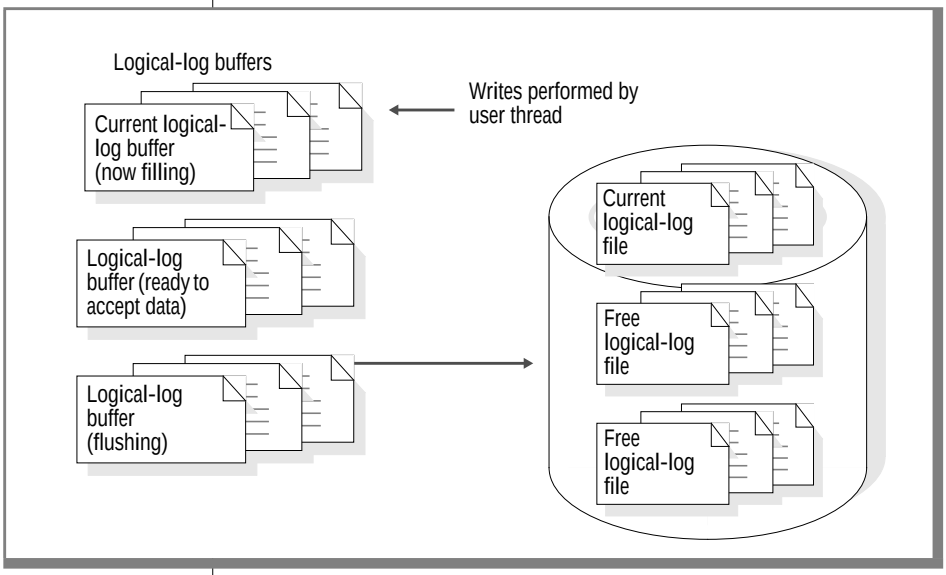


Figure 11-5
The Logical-Log Buffer and Its Relation to the Logical-Log Files on Disk

For a description of how the database server flushes the logical-log buffer, refer to "Flushing the Logical-Log Buffer" on page 11-53.

The LOGBUFF parameter in the ONCONFIG file 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). For the possible values that you can assign to this parameter, refer to "LOGBUFF" on page 33-41.

For information on the impact of TEXT and BYTE data on shared memory buffers, refer to "Buffering TEXT and BYTE Data Types" on page 11-63

Physical-Log Buffer

The database server uses the shared-memory physical-log buffer to hold before-images of dbspace pages that are going to be updated. The before-images in the physical log 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 11-47.

For information on monitoring the physical-log file, refer to "Monitoring the Physical-Log File" on page 29-47.

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. For more information on this parameter, refer to "PHYSBUFF" on page 33-69.

High-Availability Data-Replication Buffer

High-availability data replication requires two instances of the database server, a primary instance and a secondary instance, running on two computers. If you implement high-availability data replication for your database server, the database server holds logical-log records in the high-availability data-replication buffers before it sends them to the secondary database server. A high-availability 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 11-22. For more information on how the data-replication buffer is used, refer to "How Does High-Availability Data Replication Work?" on page 25-8.

The 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.

How the Database Server Manages 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 first looks 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.

How to Specify the Size of the Virtual Portion of Shared Memory

You specify the initial size of the virtual shared-memory portion by setting the SHMVIRTSIZE parameter in the ONCONFIG file. You can specify the size of segments that are later added to the virtual portion of shared memory by setting the SHMADD parameter in the ONCONFIG file.

For more information on determining the size of virtual shared memory, refer to "SHMVIRTSIZE" on page 33-83, "SHMADD" on page 33-80, and "Adding a Segment to the Virtual Portion of Shared Memory" on page 12-15.

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)
- Dictionary cache
- Stored procedures cache
- Sorting pool
- Global pool

Shared-Memory Internal Tables

The database server shared memory contains nine internal tables that track shared-memory resources. Three of these nine tables are paired with hash tables. The shared-memory internal tables are as follows:

- Buffer table and associated hash table
- Chunk table
- Dbspace table
- Lock table and associated hash table
- Page-cleaner table
- Tblspace table and associated hash table
- Transaction table
- User table

Hashing is a technique that permits rapid lookup in tables where items are added unpredictably. Three shared-memory tables have an associated hash table: the lock table, the active tblspace table, and the buffer table. These three hash tables also reside in the virtual portion of shared memory.

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 "What Is a Page?" on page 13-10.

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 this data can be overwritten. A modified, or 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 how to monitor the buffers, refer to "Monitoring Buffers" on page 29-20. For information on the database server buffers, refer to "Regular Buffers" on page 11-19.

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 two that is less than the value of BUFFERS.

Chunk Table

The chunk table tracks all chunks in the database server. If mirroring has been enabled, a corresponding mirrored chunk table is also created when shared memory is initialized. The mirrored chunk table tracks all mirrored chunks.

The chunk table in shared memory contains information that enables the database server to locate chunks on disk. This information includes the chunk number and the number of the next chunk in the dbspace. Flags also describe chunk status: mirror or primary; off-line, on-line, or recovery mode; and whether this chunk is part of a blobspace. For information on monitoring chunks, refer to "Monitoring Chunks" on page 29-53.

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. Consult your operating-system manuals for details.

Dbspace Table

The dbspace table tracks dbspaces and blobspaces 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 space is a blobspace, flags indicate the media where the blobspace is located magnetic, removable media, or optical.

For information on monitoring dbspaces, refer to "Monitoring the Database Server for Disabling I/O Errors" on page 29-52.

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. Each entry is one lock. A single transaction can own multiple locks. For an explanation of locking and the SQL statements associated with locking, refer to the *Informix Guide to SQL: Tutorial*.

The following information, which is stored in the 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 and rowid that is locked
- The tblspace where the lock is placed

You specify the maximum number of entries in the lock table by setting the LOCKS configuration parameter. For information on specifying the number of locks available to sessions, refer to "LOCKS" on page 33-39.

For information on monitoring locks, refer to "Monitoring Locks" on page 29-29.

A byte lock is generated only if you shrink the size of a data value in a VARCHAR column. The byte lock exists solely for rollforward and rollback execution, so a byte lock is created only if you are working in a database that uses logging. Byte locks appear in **onstat -k** output only if you are using row-level locking; otherwise, they are merged with the page lock.

The lock table includes an associated hash table. The number of entries in the lock hash table is based on the number of entries in the locks table. The maximum number of hash values is the largest power of two that is less than the value specified by the expression (LOCKS divided by 16).

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 parameter in the ONCONFIG file. For advice on how many page-cleaner threads to specify, refer to "CLEANERS" on page 33-14.

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, refer to the **-F** option in "onstat: Monitor Database Server Operation" on page 35-62.

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. (Do not confuse the shared-memory active tblspace table with the tblspace tblspace.) For information on monitoring tblspaces, refer to "Monitoring Tblspaces and Extents" on page 29-60.

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 displayed by **onstat -x**, see"Monitoring Transactions" on page 29-41.

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 *Informix Guide to SQL: Tutorial*, the *Informix Guide to SQL: Reference*, and the *Informix Guide to SQL: Syntax*.

UNIX

The transaction table also specifically supports the X/Open environment. Support for the X/Open environment requires INFORMIX-TP/XA. For a description of a transaction in this environment, refer to the product documentation for INFORMIX-TP/XA. ♦

User Table

The user table tracks all user threads. Each client session has one primary thread and zero to many secondary threads, depending on the level of parallelism specified. Other threads include one to monitor and control checkpoints, one to process **onmode** commands, the btree cleaner thread, and one too many 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 or to read into shared memory TEXT or BYTE DATA that is stored in a dbspace. After disk pages are read into the big buffer, they are immediately copied to regular buffers in the buffer pools. The database server also uses big buffers in sorted writes and in chunk writes during checkpoints. For information on monitoring the use of big buffers, refer to "onstat: Monitor Database Server Operation" on page 35-62.

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* (SCB). 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 structures 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* (TCB) 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 9-14.

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 9-15. For information on how to monitor the size of the stack for a session, refer to "Monitoring Sessions and Threads" on page 29-35.

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" on page 33-85.

You can alter the size of the stack for the primary thread of a specific session by setting 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 *Informix Guide to SQL: Reference*.

You can more safely alter the size of stack space by using the **INFORMIXSTACKSIZE** environment variable than by altering 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 also 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.

Dictionary Cache

When a session executes an SQL statement that requires accessing a system catalog table, the database server reads the system catalog tables and stores them in structures that it can access more efficiently. These structures are created in the virtual portion of shared memory for use by all sessions. These structures constitute the dictionary cache.

The size of the dictionary cache is not configurable.

Sorting Memory

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.

To calculate the amount of virtual shared memory that the database server might need for sorting, estimate the maximum number of sorts that might occur concurrently and multiply that number by the average number of rows times the average row size. For example, if you estimate that 30 sorts might occur concurrently, the average row size is 200 bytes, and the average number of rows in a table is 400, you can estimate the amount of shared memory that the database server needs for sorting as follows:

```
30 sorts * 200 bytes * 400 rows = 2,400,000 bytes
```

If PDQ priority is greater than 0, the maximum amount of shared memory that the database server allocates for a sort is controlled by the memory grant manager (MGM). The database server calculates the unit (quantum) of memory allocation with the following formula:

```
quantum = DS_TOTAL_MEMORY/DS_MAX_QUERIES
```

When MGM controls the resources for a query, sorting does not use more than the amount of memory specified by the DS_TOTAL_MEMORY configuration parameter. If the query requests more sorts than can concurrently fit into the DS_TOTAL_MEMORY amount of memory, some sorts must wait until memory is available. For more information about MGM, see your *Performance Guide*.

You set the **PSORT_NPROCS** environment variable to request a parallel sort. If PDQ is used, MGM divides the allocated sort memory evenly among the sort threads for the query. For more information on parallel sorts and the **PSORT_NPROCS** environment variable, refer to your *Performance Guide*.

If PDQ priority is 0, the maximum amount of shared memory that the database server allocates for a sort is about 128 kilobytes.

Stored Procedures Cache

When a session needs to access a stored procedure for the first time, the database server reads the stored procedure from the system catalog tables. The database server converts the stored procedure into executable format and stores the procedure in a cache, where it can be accessed by any session.

The size of the stored procedure cache is not configurable.

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 here and process them.

UNIX

The Communications Portion of Shared Memory

The database server allocates memory for the IPC communication portion of shared memory if and only if you configure at least one of your connections as an IPC shared-memory connection. The database server performs this allocation when you initialize 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" on page 11-11.

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 latches, also called *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 in the output from the monitoring tools), refer to "Monitoring Latches" on page 29-27.

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.

For information on how to monitor the use of locks, refer to "Monitoring Locks" on page 29-29.

For detailed information about locking and process isolation during SQL processing, see the *Informix Guide to SQL: Tutorial*. For further information about locking and shared memory, refer to "Shared-Memory Buffer Locks".

The 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 and none intends to modify the data.

The 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.

For more information about locking, see "Lock Table" on page 11-29 and "Shared-Memory Buffer Locks".

How Database Server Threads Access Shared Buffers

The database server threads access shared buffers through a system of queues, using latches and locks to synchronize access and protect data.

LRU Queues

Each regular buffer is tracked through several linked lists of pointers to the buffer table. A regular buffer holds data for the purpose of caching. These linked lists are the least-recently used (LRU) queues.

The LRUS parameter in the ONCONFIG file specifies the number of LRU queues to create when the database server shared memory is initialized. You can tune the value of LRUS, combined with the LRU_MIN_DIRTY and LRU_MAX_DIRTY parameters, to control how frequently the shared-memory buffers are flushed to disk.

LRU Queue Components

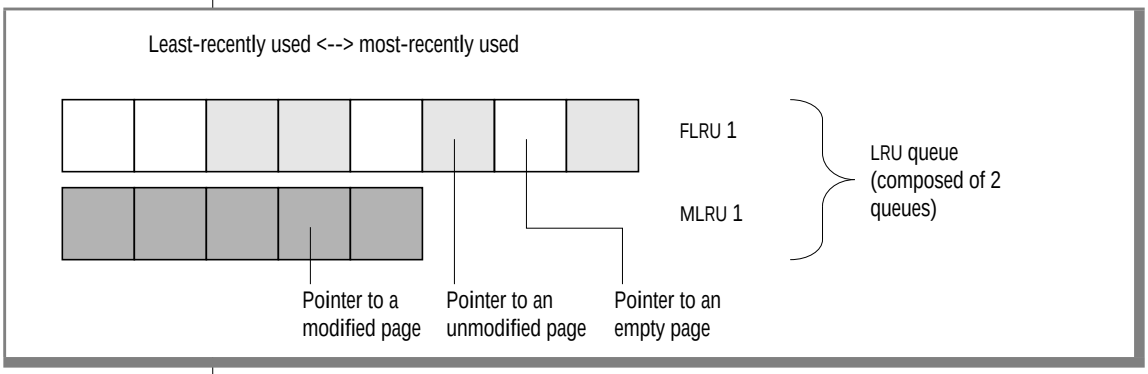
The LRU queue is composed of two queues, the FLRU and the MLRU queues.

Each LRU queue is actually a pair of linked lists, as follows:

- One list tracks free or unmodified pages in the queue.
- One list 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 11-6 on page 11-38 illustrates the structure of the LRU queues.

Figure 11-6
LRU Queue



Why Are Pages Ordered 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. The number of buffers in each queue is calculated by dividing the total number of buffers (BUFFERS) by the number of LRU queues (LRUS).

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 29-23.

How Many LRU Queues Should You 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.

Informix recommends initial values for the LRUS configuration parameter based on the number of CPUs that are available on your computer. If your computer is a uniprocessor, start by setting LRUS to 4. If your computer is a multiprocessor, use the following formula:

```
LRUS = max(4, (no. of cpu VPs))
```

After you give an initial value to LRUS, monitor your LRU queues with **onstat -R**. If you find that the percent of dirty LRU queues consistently exceeds the value of the LRU_MAX_DIRTY parameter, add more LRU queues by increasing the value of the LRUS configuration parameter.

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 configuration parameter. By increasing 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.

How Many Cleaners Should You Allocate?

In general, Informix recommends that you 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 can occur if you do not have enough page-cleaner threads allocated. The percent of dirty pages might exceed 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 much more dirty 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 in cleaning an additional 4 queues. By increasing the number of CLEANERS to 12, each of the queues, now shortened, 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 have a sufficient number of page cleaners configured, checkpoints take longer, causing overall performance to suffer.

Limiting the 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 LRU_MAX_DIRTY configuration parameter.

By specifying when page cleaning begins, the LRU_MAX_DIRTY configuration parameter limits the number of page buffers that can be appended to an MLRU queue. The initial setting of LRU_MAX_DIRTY is 60, 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 "How the Database Server Flushes Data to Disk" on page 11-46.

Figure 11-7 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

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 11-7 *How LRU_MAX_DIRTY Initiates Page Cleaning to Limit the Size of the MLRU Queue*

When MLRU Cleaning Ends

You can also specify the point at which MLRU cleaning can end. The LRU_MIN_DIRTY configuration parameter specifies the acceptable percent of buffers in an MLRU queue. The initial setting of LRU_MIN_DIRTY is 50, meaning that page cleaning is no longer 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 11-8 on page 11-42 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

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 .050 = 500
```

Figure 11-8
How LRU_MIN_DIRTY Specifies the Point at Which Page Cleaning Can End

For more information on how the database server flushes the buffer pool, refer to "How the Database Server Flushes Data to Disk" on page 11-46.

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 when the database server does a read-ahead.

The RA_THRESHOLD parameter specifies the number of unprocessed pages in memory that cause the database server to do 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 an example of the output that the **onstat -p** command produces to enable you to monitor the database server use of read-ahead, refer to "Monitoring Shared-Memory Profile" on page 29-19 and "-p Option" on page 35-84 under the heading "onstat: Monitor Database Server Operation."

How a Database Server Thread Accesses a Buffer Page

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 11-36.)

The process of accessing a data buffer consists of the following steps:

1. Identify the data requested by physical page number.
2. Determine the level of lock access needed by the thread for the requested buffer.
3. Attempt to locate the page in shared memory.
4. If the page is not in shared memory, locate a buffer in an FLRU queue, and read the page in from disk. If the page is in shared memory, proceed with step 5.
5. Proceed with processing, locking the buffer if necessary.
6. When finished accessing the buffer, release the lock.
7. Wake waiting threads with compatible lock-access types, if any exist.

Identify the Page

The database server threads request a specific data row, and the database server searches for the page that contains the row.

Determine the Level of Lock Access

Next the database server determines the requested level of lock access: share or exclusive.

Try to Locate the Page in Shared Memory

The thread first attempts to locate the requested page in shared memory. To do this, it acquires a mutex on the hash table that is associated with the buffer table. Then, it searches the hash table to see if an entry matches the requested page. If the thread finds an entry for the page, it releases the mutex on the hash table and tries to acquire the mutex on the buffer entry in the buffer table.

The thread tests the current lock-access level of the buffer. If the levels are compatible, the requesting thread gains access to the buffer and sets its own lock. If the current lock-access level is incompatible, the requesting thread puts itself on the wait queue for the buffer.

The buffer state, unmodified or modified, is irrelevant to locking; even unmodified buffers can be locked.

If you configure the database server to use read-ahead, the database server performs a read-ahead request when the number of pages specified by the RA_THRESHOLD parameter remains to be processed in memory.

Locate a Buffer and Read Page from Disk

If the requested page must be read from disk, the thread first locates a usable buffer in the FLRU queues. The database server selects an FLRU queue at random and tries to acquire the mutex associated with the queue. If the mutex can be acquired, the buffer at the least-recently used end of the queue is used. If another thread holds the mutex, the first thread tries to acquire the mutex of another FLRU queue.

If you configure the database server to use read-ahead, the database server reads the number of pages specified by the RA_PAGES configuration parameter.

Lock the Buffer If Necessary

After a usable buffer is found, the buffer is temporarily removed from the FLRU queue. The thread creates an entry in the shared-memory buffer table as the page is read from disk into the buffer.

Release the Buffer Lock and Wake a Waiting Thread

When the thread is finished with the buffer, it releases the buffer lock. If any threads are waiting for the buffer, it wakes one up. However, this procedure varies, depending on whether the releasing thread modified the buffer.

When the Buffer Is Not Modified

If a thread does not modify the data, it releases the buffer as unmodified.

The release of the buffer occurs in steps. First, the releasing thread acquires the mutex on the buffer table that enables it to modify the buffer entry.

Next, it checks if other threads are sleeping, waiting for this buffer. If so, the releasing thread wakes the first thread in the wait queue that has a compatible lock-access type. The waiting threads are queued according to priorities that encompass more than just *first-come, first-served* hierarchies. (Otherwise, for example, threads waiting for exclusive access could wait forever.)

If no thread in the wait queue has a compatible lock-access type, any thread waiting for that buffer can receive access.

If no thread is waiting for the buffer, the releasing thread tries to release the buffer to the FLRU queue where it was found. If the latch for that FLRU queue is unavailable, the thread tries to acquire a latch for a randomly selected FLRU queue. When the FLRU queue latch is acquired, the unmodified buffer is linked to the most-recently used end of the queue.

After the buffer is returned to the FLRU queue, or the next thread in the wait queue is awakened, the releasing thread removes itself from the user list for the buffer and decrements the shared-user count by one.

When the Buffer Is Modified

If the thread intends to modify the buffer, to update a row in a table, for example, it acquires the mutex for the buffer and changes the buffer lock-access type to exclusive.

In most cases, a copy of the before-image of the page is needed for data consistency. If necessary, the thread determines whether a before-image of this page was written to either the physical-log buffer or the physical log since the last checkpoint. If not, a copy of the page is written to the physical-log buffer. Then the data in the page buffer is modified. If any transaction records are required for logging, those records are written to the logical-log buffer.

After the mutex for the buffer is released, the thread is ready to release the buffer. First, the releasing thread acquires the mutex on the buffer table that enables it to modify the buffer entry. Next, the releasing thread updates the time stamp in the buffer header so that the time stamp on the buffer page and the time stamp in the header match. Statistics describing the number and types of writes performed by this thread are updated.

The lock is released as described in the previous section, but the buffer is appended to the MLRU queue associated with the original FLRU queue.

How the Database Server Flushes 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.

Buffer flushing is managed by the page-cleaner threads. 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 "CLEANERS" on page 33-14.

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.

Events That Prompt Flushing of the Regular Buffers

Flushing of the regular buffers is initiated by any one of the following three conditions:

- The number of buffers in an MLRU queue reaches the number specified by LRU_MAX_DIRTY.
- 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.

Flushing Before-Images First

The overriding rule of buffer flushing is that 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 regular 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 regular 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 disk pages in the physical-log buffer. Before a disk page can be modified, a before-image of the disk page must already be stored in the physical log. If the before-image has been written to the physical-log buffer but not to the physical log on disk, the physical-log buffer must be flushed to disk before the modified page can be flushed to disk. This action is required for the fast-recovery feature. Writing the before-image to the physical log buffer and then flushing the buffer page to disk is illustrated in Figure 11-9 on page 11-48.

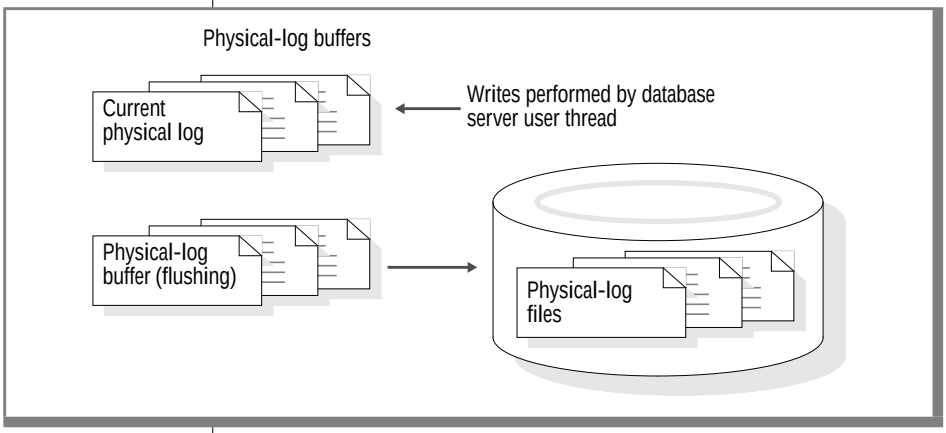


Figure 11-9
The Physical-Log Buffer and its Relation to the Physical Log on Disk

Both the physical-log buffer and the physical log contribute toward maintaining the physical and logical consistency of the data. For a description of physical logging, refer to Chapter 20, "What Is Physical Logging?" For a description of fast recovery, refer to Chapter 22, "What Is Fast Recovery?"

Events That Prompt Flushing of the Physical-Log Buffer

The following three events cause the current physical-log buffer to flush:

- The current physical-log buffer becomes full.
- A modified page in shared memory must be flushed, but the before-image is still in the current physical-log buffer.
- A checkpoint occurs.

The contents of the physical-log buffer must always be flushed to disk before any data buffers. This rule is required for the fast-recovery feature.

The database server uses only one of the two physical-log buffers at a time. This buffer is the current physical-log buffer. Before the database server flushes the current physical-log buffer to disk, it makes the other buffer the current buffer so that it can continue writing while the first buffer is being flushed.

When the Physical-Log Buffer Becomes Full

Buffer flushing that results from the physical-log buffer becoming full proceeds as follows.

When a user thread needs to write a before-image to the physical-log buffer, it acquires the mutex associated with the physical-log buffer and the mutex associated with the physical log on disk. If another thread is writing to the buffer, the incoming thread must wait for the mutexes to be released.

After the incoming thread acquires the mutexes, but before the write, the thread checks to see what percent of the physical log is full.

If the Log Is More Than 75 Percent Full

If the log is more than 75 percent full, the thread sets a flag to request a checkpoint. Next, the thread claims the amount of space in the buffer that it needs for its write and releases the buffer mutex so that other threads can access the buffer. Finally, it copies the data into the space that it claimed in the buffer. The checkpoint does not begin until all user threads, including this one, are out of critical sections. For a description of a critical section, refer to "Critical Sections" on page 11-56.

If the Log Is Less Than 75 Percent Full

If the log is less than 75 percent full, the thread compares the page counter in the physical-log buffer header to the buffer capacity. If this one-page write does not fill the physical-log buffer, the thread reserves space in the log buffer for the write and releases the mutex. Any thread waiting to write to the buffer is awakened. After the thread releases the mutex, it writes the page to the reserved space in the physical-log buffer. The sequence of this operation increases concurrency and eliminates the need to hold the mutex during the write.

If this one-page write fills the physical-log buffer, flushing is initiated. First the page is written to the current physical-log buffer, filling it. Next, the thread latches the other physical-log buffer. The thread switches the shared-memory current-buffer pointer, making the newly latched buffer the current buffer. The mutex on the physical log on disk and the mutex on this new, current buffer are released, which permits other user threads to begin writing to the new current buffer. Last, the full buffer is flushed to disk, and the mutex on the buffer is released.

Each write to the physical-log buffer writes one page.

How the Database Server Synchronizes Buffer Flushing

When shared memory is first initialized, all buffers are empty. As processing occurs, data pages are read from disk into the buffers, and user threads begin to modify these pages.

Ensuring That Physical-Log Buffers Are Flushed First

When page cleaning is initiated on the shared-memory buffer pool, the page-cleaner thread must coordinate the flushing so that the physical-log buffer is flushed first. Time-stamp comparison determines the order.

The database server stores a time stamp each time that the physical-log buffer is flushed. If a page-cleaner thread needs to flush a page in a shared-memory buffer, the page cleaner compares the time stamp in the modified buffer with the time stamp that indicates the point when the physical-log buffer was last flushed.

If the time stamp on the page in the buffer pool is equal to or more recent than the time stamp for the physical-log buffer flush, the before-image of this page conceivably could be contained in the physical-log buffer. In this case, the physical-log buffer must be flushed before the shared-memory buffer pages are flushed.

After the physical-log buffer is flushed, the user thread updates the time stamp in shared memory that describes the most-recent physical-log buffer flush. The specific page in the shared-memory buffer pool that is marked for flushing is now flushed. The number of modified buffers in the queue is compared to the value of LRU_MIN_DIRTY. If the number of modified buffers is greater than the value represented by LRU_MIN_DIRTY, another page buffer is marked for flushing. The time-stamp comparison is repeated. If required, the physical-log buffer is flushed again.

When no more buffer flushing is required, the page-cleaner threads sleep *forever*, which means they sleep until buffer flushing is required again, and they are awakened to do the work. (For more information, refer to "Sleep Queues" on page 9-17.) You can tune the page-cleaning parameters (LRU_MIN_DIRTY and LRU_MAX_DIRTY) to influence the frequency of buffer flushing. For a description of how these parameters determine when page cleaning begins and ends, refer to "LRU Queues" on page 11-37.

How Write Types Describe Flushing Activity

The database server provides you with information about the specific condition that prompted buffer-flushing activity by defining three types of writes and counting 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 on page 35-75.

If you implement mirroring for the database server, data is always written to the primary chunk first; then the write is repeated on the mirrored chunk. Writes to a mirrored 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 29-23.

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 "LRU Queues" on page 11-37.)

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 by either increasing the number of page cleaners or decreasing the value of LRU_MAX_DIRTY.

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 you that specified in the LRU_MAX_DIRTY 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 tapes. For a description of logical logging, refer to Chapter 18, "What Is the Logical Log?"

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.

Each of these events is discussed in detail in the following sections.

When the Logical-Log Buffer Becomes Full

When a user thread needs to write records to the logical-log buffer, it acquires the mutexes associated with the logical-log buffer and the current logical log on disk. If another thread is writing to the buffer, the incoming thread must wait for the mutexes to be released.

After the incoming thread acquires the mutexes, but before the write, the thread checks how much logical-log space is available on disk. When the logical-log space on disk is full, and the database server switches to a new logical log, it checks if the percent of used log space is greater than the long-transaction high-water mark, specified by the LTXHWM parameter in the ONCONFIG file. For a description of this parameter and for information on specifying a value for it, refer to "LTXHWM" on page 33-51.

If no long-transaction condition exists, the logical-log I/O thread compares the available space in the logical-log buffer with the size of the record to be written. If the write does not fill the logical-log buffer, the thread writes the record, releases latches, and awakens any threads that are waiting to write to the buffer.

If the write fills the logical-log buffer, flushing is initiated as follows:

1. The thread latches the next logical-log buffer. The thread then switches the shared-memory current-buffer pointer, making the newly latched buffer the current buffer.
2. The thread writes the new record to the new current buffer. The thread releases the latch on the logical log on disk and the latch on this new, current buffer, permitting other logical-log I/O threads to begin writing to the new current buffer.
3. The full logical-log buffer is flushed to disk, and the latch on the buffer is released. This logical-log buffer is now available for reuse.

After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging

If a transaction is prepared or terminated in a database with unbuffered logging, the logical-log buffer is immediately flushed. Flushing might cause a waste of some disk space. Typically, many logical-log records are stored on a single page. However, because the logical-log buffer is flushed in whole pages, even if only one transaction record is stored on the page, the whole page is flushed. In the worst case, a single COMMIT logical-log record (COMMIT WORK) could occupy a page on disk, and all remaining space on the page would be unused. However, the cost in disk space of using unbuffered logging is minor compared to the benefits of insured data consistency.

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 *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 11-57.

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.

How the Database Server Achieves Data Consistency

The database server uses the following three procedures to ensure that the data that is destined for disk is actually recorded intact on disk:

- Critical sections
- Checkpoints
- Time stamps

These procedures ensure that multiple, logically related writes are recorded as a unit; that data in shared memory is periodically made consistent with data on disk; and that a buffer page that is written to disk is actually written in entirety.

Critical Sections

A *critical section* is a section of code that makes a set of disk modifications that must be performed as a single unit; either all the modifications must occur, or none can occur.

A thread that is in a critical section is holding shared-memory resources. Within the space of the critical section, the database server cannot determine which shared-memory resources should be released and which changes should be undone to return all data to a consistent point. Therefore, if a virtual processor is terminated while a thread is in a critical section, the database server takes the two following steps to ensure that all data is returned to the last known point of consistency:

- The database server aborts immediately.
- The database server initiates fast recovery the next time that it is initialized.

Fast recovery is the procedure that the database server uses to restore the physical and logical consistency of data quickly, up to and including the last record in the logical log. For a description of fast recovery, refer to Chapter 22, "What Is Fast Recovery?"

Checkpoints

The term *checkpoint* refers to the point in the database server operation when the pages on disk are synchronized with the pages in the shared-memory buffer pool. When a checkpoint completes, all physical operations are complete, the MLRU queue is empty, and the database server is said to be physically consistent.

Events That Initiate a Checkpoint

Any user thread can initiate a check to determine if a checkpoint is needed. A checkpoint is initiated under any one of five conditions:

- The checkpoint interval, specified by the configuration parameter CKPTINTVL, has elapsed, and one or more modifications have occurred since the last checkpoint.
- The physical log on disk becomes 75 percent full.
- The database server detects that the next logical-log file to become current contains the most-recent checkpoint record.
- Certain administrative tasks, such as adding a chunk or a dbspace, take place.
- The database server administrator initiates a checkpoint from the command line with **onmode -c**

UNIX

- The database server administrator initiates a checkpoint from the ON-Monitor, Force-Ckpt menu. ♦

An administrator might initiate a checkpoint to force a new checkpoint record in the logical log. Forcing a checkpoint would be a step in freeing 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).

The following section outlines the main events that occur once a user thread raises the checkpoint-requested flag. The following events occur during a checkpoint:

1. The database server prevents user threads from entering critical sections.
2. The **main_loop()** thread flushes the physical-log buffer.
3. The page-cleaner threads flush modified pages in the buffer pool to disk. Flushing is performed as a chunk write.
4. The **main_loop()** thread writes a checkpoint record to the logical-log buffer.
5. The physical log on disk is logically emptied (current entries can be overwritten).
6. The logical-log buffer is flushed to the current logical-log file on disk.
7. The **main_loop()** thread updates configuration and dbspace backup information to reserved pages.

User Threads Cannot Enter a Critical Section

Once the checkpoint-requested flag is set, user threads are prevented from entering portions of code that are considered critical sections. User threads that are within critical sections of code are permitted to continue processing to the end of the critical sections.

The main_loop() Thread Flushes the Physical-Log Buffer

After all threads have exited from critical sections, the **main_loop()** thread resets the shared-memory pointer from the current physical-log buffer to the other buffer and flushes the buffer. After the buffer is flushed, the page-cleaner thread updates the time stamp that indicates the most-recent point at which the physical-log buffer was flushed.

Page-Cleaner Threads Flush Modified Pages in the Buffer Pool

Next, the page cleaners flush all modified pages in the shared-memory buffer pool. This flushing is performed as a chunk write.

The main_loop() Thread Writes Checkpoint Record

After the modified pages have been written to disk, the page-cleaner thread writes a *checkpoint-complete* record in the logical-log buffer.

Physical Log Is Logically Emptied

After the checkpoint-complete record is written to disk, the physical log is logically emptied, meaning that current entries in the physical log can be overwritten.

Logical-Log Buffer Is Flushed to the Logical-Log File on Disk

Next, the logical-log buffer is flushed to the logical-log file on disk.

The main_loop Thread Updates Reserved Pages

The **main_loop()** thread next begins writing all configuration and dbspace backup information to the appropriate reserved pages, regardless of whether changes have occurred since the last checkpoint.

When dbspaces, primary chunks, or mirrored chunks are added or dropped from the database server, the changes are recorded in descriptor tables in shared memory. If changes occurred since the last checkpoint, the **main_loop()** thread writes the descriptor tables from shared memory to the appropriate reserved page in the root dbspace. Otherwise, the **main_loop()** thread ignores the reserved pages that describe the dbspaces, primary chunks, and mirrored chunks. The **main_loop()** thread writes all checkpoint statistics to the appropriate reserved page in the root dbspace.

Checkpoint Is Critical to Fast Recovery

The database server generates at least one checkpoint for each span of the logical-log space to guarantee that it has a checkpoint at which to begin fast recovery.

As fast recovery begins, the database server brings data to physical consistency as of the last checkpoint by restoring the contents of the physical log.

During the next stage of fast recovery, the database server reprocesses the transactions contained in the logical logs, beginning at the point of the last checkpoint record and continuing through all the records contained in the subsequent logical logs.

After fast recovery completes, the database server data is consistent up through the last completed transaction. That is, all committed transactions recorded in the logical logs on disk are retained; all incomplete transactions (transactions with no COMMIT WORK entry in the logical logs on disk) are rolled back.

Time Stamps

The database server uses a time stamp to identify a time when an event occurred relative to other events of the same kind. The time stamp is not a literal time that refers to a specific hour, minute, or second. It is a 4-byte integer that the database server assigns sequentially. When the database server compares two time stamps, its algorithm accounts for the possibility that wraparound has occurred.

Time Stamps on Disk Pages

Each disk page has one time stamp in the page header and a second time stamp in the last 4 bytes on the page. The page-header and page-ending time stamps are synchronized after each write, so they should be identical when the page is read from disk. Each read compares the time stamps as a test for data consistency. If the test fails, an error is returned to the user thread, indicating either that the disk page was not fully written to disk or that the page has been partially overwritten on disk or in shared memory. For a description of the content of a dbspace page, refer to "Structure and Storage of a Dbspace Page" on page 38-34.

Time Stamps on Blobpages

In addition to the page-header and page-ending time-stamp pair, each disk page that contains TEXT or BYTE data also contains one member of a second pair of time stamps. This second pair of time stamps is the blob time-stamp pair. The time stamp that appears on the disk page where the TEXT or BYTE data is stored is paired with a time stamp that is stored with the forward pointer to this data segment. The forward pointer is stored either in the data row (with the descriptor) or with the previous segment of TEXT or BYTE data. For more information on time stamps on blobpages, refer to "Blobspace Structure and Storage" on page 38-59.

A blob time-stamp pair is updated whenever a TEXT or BYTE column is updated. When TEXT or BYTE data in a row is updated, the new object is stored on disk, and the forward pointer that is stored with the descriptor is revised to point to the new location. The blob time stamp in the data row is updated and synchronized with the blob time stamp on the disk page of the new TEXT or BYTE object.

Blob Time Stamps with Dirty Read and Committed Read Isolation Levels

Because TEXT or BYTE objects can be very large, it might be impossible to retrieve the data simultaneously with the rest of the row. Coordination is needed for TEXT or BYTE reads at the Dirty Read or Committed Read level of isolation. Therefore, each read compares the two members of the blob time-stamp pair as a test for logical consistency of data. If the two time stamps in the pair differ, this inconsistency is reported as a part of consistency checking. The error indicates either that the pages have been corrupted or that the forward pointer is no longer valid.

To understand how a forward pointer for TEXT or BYTE data might become invalid, consider the following examples.

Dirty Read

A program using Dirty Read isolation is able to read rows that have been deleted, provided the deletion has not yet been committed. Assume that a user thread is deleting TEXT or BYTE data from a row. During the delete process, another user thread that is operating with a Dirty Read isolation level reads the same row, searching for the blob-descriptor information. In the meantime, the first transaction completes, the TEXT or BYTE data is deleted, the space is freed, and a third thread starts to write new TEXT or BYTE data in the newly freed space where the first TEXT or BYTE object was stored. Eventually, when the second user thread starts to read the data at the location where the first object was stored, the thread compares the time stamp from the descriptor with the time stamp that precedes the data. The time stamps do not match. The time stamp on the blobpage is greater than the time stamp in the forward pointer, indicating to the user thread that the forward pointer information is obsolete.

Committed Read

If a program is using Committed Read isolation, the problem just described cannot occur because the database server does not see a row that has been marked for deletion. However, under Committed Read, no lock is placed on an undeleted row when it is read. BYTE or TEXT data is read in a second step, after the row is fetched. During this step, another program could delete the row, commit the deletion, and reuse the space on the disk page. If the space is reused in the interim, the time stamp for the TEXT or BYTE data is greater than the time stamp in the forward pointer. In this case, the comparison indicates the obsolete pointer information, and the inconsistency is reported.

Buffering TEXT and BYTE Data Types

You can assign BYTE and TEXT data types, or columns, to either dbspaces or blobspaces.

Writing TEXT and BYTE Data

The database server writes TEXT and BYTE data to disk pages in a dbspace in the same way that it writes any other data type. For more information, refer to "How the Database Server Flushes Data to Disk" on page 11-46.

The database server writes TEXT and BYTE data to a blobspace differently than 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 "Blobspace Structure and Storage" on page 38-59.

Blobpages Do Not Pass Through 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 has the potential to 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 "Blobspace Logging" on page 18-27.

Blobpages stored on optical media are not written to dbspace and logical-log backup tapes due to the high reliability of optical media.

TEXT and BYTE Objects Are Created Before the Data Row Is Inserted

When TEXT or BYTE data is written to disk, the row to which it belongs might not exist yet. During an insert, for example, the TEXT or BYTE data is transferred before the rest of the row data. After the TEXT or BYTE object is stored, the data row is created with a 56-byte descriptor that points to its location. For a description of how TEXT and BYTE data types are stored physically, refer to "TEXT and BYTE Data Storage and the Descriptor" on page 38-61.

Blobpage Buffers Are Created for the Duration of the Write

To receive TEXT or BYTE 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 blob buffers and, therefore, can access only one TEXT or BYTE object at a time.

TEXT or BYTE 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 TEXT or BYTE data is written to disk, the database server deallocates the pair of blobspace buffers. Figure 11-10 illustrates the process of writing TEXT or BYTE data to a blobspace.

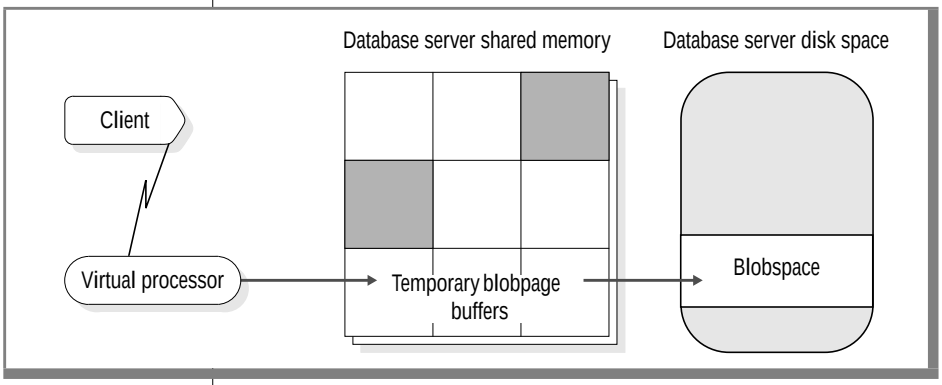


Figure 11-10
Writing TEXT or BYTE data to a Blobspace

Tracking Blobpages

Blobspace blobpages are allocated and tracked using 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.

Chapter

12

Managing Shared Memory

Setting Operating-System Shared-Memory Configuration Parameters 12-3
Maximum Operating-System Shared-Memory Segment Size . . . 12-4
Maximum Number of Shared-Memory Identifiers. 12-4
Shared-Memory Lower-Boundary Address 12-5
Maximum Amount of Shared Memory for One Process 12-5
Semaphores 12-6

Setting Database Server Shared-Memory Configuration Parameters 12-6
Setting Parameters for Resident Shared Memory with ON-Monitor. 12-7
Setting Configuration Parameters for Resident Shared Memory with a Text Editor. 12-8
Setting Parameters for Virtual Shared Memory with ON-Monitor. 12-10
Setting Parameters for Virtual Shared Memory with a Text Editor 12-11
Setting Parameters for Shared-Memory Performance Options with ON-Monitor 12-12
Setting Parameters for Shared-Memory Performance Options with a Text Editor 12-13

Reinitializing Shared Memory 12-14

Turning Residency On or Off for Resident Shared Memory 12-14
Turning Residency On or Off in On-Line Mode. 12-14
Turning Residency On or Off for the Next Time You Reinitialize Shared Memory 12-15

Adding a Segment to the Virtual Portion of Shared Memory 12-15

Forcing a Checkpoint 12-16

This chapter tells you how to perform tasks related to managing the use of shared memory with the database server. It assumes you are familiar with the terms and concepts contained in Chapter 11, "Shared Memory."

This chapter describes how to perform the following tasks:

- Set the shared-memory configuration parameters
- Reinitialize shared memory
- Turn residency on or off for the resident portion of the database server shared memory
- Add a segment to the virtual portion of shared memory
- Force a checkpoint

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 *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. These parameters are described by function in the following list. Parameter names are not provided because names vary among platforms, and not all parameters exist on all platforms:

- 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

UNIX

- Shared-memory lower-boundary address
- Maximum number of attached shared-memory segments per process
- Maximum amount of shared memory system wide
- Maximum number of semaphore identifiers
- Maximum number of semaphores
- Maximum number of semaphores per identifier

For specific information about your operating-system environment, refer to the machine-notes file that is provided with the database server. For more information about the machine-notes file, refer to "Documentation Notes, Release Notes, Machine Notes" on page 15 in the Introduction. ♦

Maximum Operating-System 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 is too large; that is, if the segment size is greater than 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.

Maximum Number of Shared-Memory Identifiers

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 11-12.

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.

Shared-Memory Lower-Boundary Address

When the database server attaches shared-memory segments subsequent to the first segment, it assumes that the segment can be attached contiguous with the previous one—that is, that a segment can be attached at the address of the previous segment plus the size of that segment. However, your operating system might set a parameter that defines a lower-boundary address for attaching shared-memory segments. If the size of a segment would cause it to cross the lower-boundary address, the segment is attached at a point beyond the end of the previous segment, creating a gap between shared-memory segments. For an illustration of this problem, refer to "How Virtual Processors Attach to Shared Memory" on page 11-12.

Maximum Amount of Shared Memory for One Process

Check that the maximum amount of memory that can be allocated for one process is equal to the total addressable shared-memory size for a single operating-system process. The following equation expresses the concept another way:

```
Maximum amount of shared memory for one process =
    (Maximum number of attached shared-memory segments per
    process) x (Maximum shared-memory segment size)
```

If this relationship does not hold, one of two undesirable situations could develop:

- If the total amount of shared memory is less than the total addressable shared-memory size, you are able to address more shared memory for the operating system than is available.
- If the total amount of shared memory is greater than the total addressable size of shared memory, you can never address some amount of shared memory that is available. That is, space that could potentially be used as shared memory cannot be allocated.

UNIX

Semaphores

The database server operation requires 1 UNIX semaphore for each virtual processor, 1 for each user who connects to the database server through shared memory (ipcshm protocol), 6 for database server utilities, and 16 for other purposes.

Setting Database Server Shared-Memory Configuration Parameters

Shared-memory configuration parameters are divided 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
- Shared-memory parameters that affect performance

You can set shared-memory configuration parameters in the following ways:

UNIX

- Using ON-Monitor ♦
- Using a text editor

You must be **root** or user **informix** to use either method.

Regardless of which method you use, you must reinitialize shared memory to put the changes into effect.

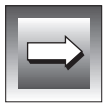
UNIX

Setting Parameters for Resident Shared Memory with ON-Monitor

To set the configuration parameters for the resident portion of shared memory using ON-Monitor, select **Parameters→Shared-Memory**. Figure 12-1 shows an example of a Shared-Memory screen. The shaded entries set configuration parameters for the resident portion of shared memory.

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                      SHARED MEMORY PARAMETERS
Server Number                    [   0]       Server Name [odyssey_ol        ]
Server Aliases [oddsoc_ol1,oddsoc_ol2,oddsoc_ol3                             ]
Dbspace Temp   [                                                             ]
Deadlock Timeout                 [   0] Secs  Number of Page Cleaners   [   1]
Forced Residency                 [Y]          Stack Size (Kbytes)       [  32]
Non Res. SegSize (Kbytes)    [   4000]        Optical Cache Size (Kbytes)[  0]

Physical Log Buffer Size  [   32] Kbytes     Dbspace Down Option        [  0]
Logical Log Buffer Size   [   32] Kbytes     Preserve Log for Log Backup  [N]
Max # of Logical Logs     [    6]            Transaction Timeout       [ 300]
Max # of Locks            [   2000]          Long TX HWM               [  50]
Max # of Buffers          [   200]           Long TX HWM Exclusive     [  60]
                                             Index Page Fill Factor    [  90]
                                             Add SegSize (Kbytes)  [    8192]
                                             Total Memory (Kbytes) [       0]
 Shared memory size        [       546] Kbytes        Page Size [    2] Kbytes

Enter a unique value to be associated with this version of Informix Dynamic Server
```

Figure 12-1
ON-Monitor Shared-Memory Screen



Important: *The configuration parameters SHMADD and SHMTOTAL are described with the parameters that affect the resident portion of shared memory, but they affect both the resident and virtual portions of shared memory.*

Figure 12-2 shows an example of only the Shared-Memory screen entries that affect the configuration of the resident portion of shared memory. For each entry, it shows within brackets ([ ]) the name of the associated parameter in the ONCONFIG file.

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                    SHARED MEMORY PARAMETERS
Server Number             [SERVER_NUM]


                                        Number of Page Cleaners[CLEANERS]
Forced Residency                 [Y]


Physical Log Buffer Size  [  PHYSBUFF] Kbytes
Logical Log Buffer Size   [   LOGBUFF] Kbytes
Max # of Logical Logs     [  LOGFILES]
Max # of Locks            [     LOCKS]
Max # of Buffers          [   BUFFERS]       Add SegSize (Kbytes)    [SHMADD]
                                             Total Memory (Kbytes) [SHMTOTAL]

Shared memory size        [       546] Kbytes        Page Size [    2] Kbytes

Enter a unique value to be associated with this version of Informix Dynamic Server
```

Figure 12-2 *ONCONFIG Parameter for Each of the Shared-Memory Entries*

For more information on the ONCONFIG parameters that are associated with the resident portion of shared memory, refer to Figure 12-3 on page 12-9.

Setting Parameters for Resident Shared Memory with a Text Editor

You can use a text editor to set shared-memory configuration parameters at any time. To set a shared-memory configuration parameter, use the editor to 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 reinitialize shared memory.

Figure 12-3 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. The page references in the third column refer to summary descriptions of the parameters in Chapter 33, "Configuration Parameters."

Figure 12-3
Configuring the Resident Portion of Shared Memory

ONCONFIG Parameter	Purpose	Reference
BUFFERS	Specifies the maximum number of shared-memory buffers	page 33-12
CLEANERS	Specifies the number of page-cleaner threads that the database server is to run	page 33-14
LOCKS	Specifies the maximum number of locks for database objects; for example, rows, key values, pages, and tables	page 33-39
LOGBUFF	Specifies the size of the logical-log buffers	page 33-41
PHYSBUFF	Specifies the size of the physical-log buffers	page 33-69
RESIDENT	Specifies residency for the resident portion of the database server shared memory	page 33-74
SERVERNUM	Specifies a unique identification number for the database server on the local host computer	page 33-79
SHMADD	Specifies the size of dynamically added shared-memory segments	page 33-80
SHMTOTAL	Specifies the total amount of memory to be used by the database server	page 33-82

UNIX

Setting Parameters for Virtual Shared Memory with ON-Monitor

To set the configuration parameters for the virtual portion of shared memory using ON-Monitor, select **Parameters→Shared-Memory**. Figure 12-4 shows an example of a Shared-Memory screen. The shaded entries set configuration parameters for the virtual portion of shared memory.

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                      SHARED MEMORY PARAMETERS
 Server Number                  [2   ]       Server Name [lashley_ol        ]
 Server Aliases [cole,davison,stackhouse                                    ]
 Dbspace Temp   [                                                           ]
 Deadlock Timeout               [  60] Secs  Number of Page Cleaners   [   1]
 Forced Residency                 [N]        Stack Size (Kbytes)       [  32]
 Non Res. SegSize (Kbytes)    [   4000]      Optical Cache Size (Kbytes)   [   0]

 Physical Log Buffer Size [  32] Kbytes     Dbspace Down Option         [   0]
 Logical Log Buffer Size  [  32] Kbytes     Preserve Log for Log Backup [   N]
 Max # of Logical Log     [  14]            Transaction Timeout         [ 300]
 Max # of Locks           [ 2000]           Long TX HWM                 [  50]
 Max # of Buffers         [   80]           Long TX HWM Exclusive       [  60]
                                            Index Page Fill Factor      [  90]
                                            Add SegSize (Kbytes)        [8192]
                                            Total Memory (Kbytes)       [   0]

 Shared memory size        [       528] Kbytes        Page Size [    2] Kbytes
Enter a unique value to be associated with this version of Informix Dynamic Server
```

Figure 12-4 *Setting Virtual Shared-Memory Configuration Parameters*

Figure 12-5 shows only the Shared Memory screen entries that affect the configuration of the virtual portion of shared memory. For each entry, it shows the name of the associated parameter in the ONCONFIG file within brackets ([ ]).

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                      SHARED MEMORY PARAMETERS


                                           Stack Size (Kbytes)  [STACKSIZE]
 Non Res. SegSize (Kbytes) [SHMVIRTSIZE]
```

Figure 12-5 *ONCONFIG Parameters for Each of the Virtual Shared-Memory Entries*

For more information on the ONCONFIG parameters that affect the configuration of the virtual portion of shared memory, refer to Figure 12-6.

Setting Parameters for Virtual Shared Memory with a Text Editor

You can use a text editor at any time to set the virtual shared-memory configuration parameters. To set the virtual shared-memory configuration parameters with a text editor, use the editor to locate the parameter in the file, enter the new value or values, and rewrite the file to disk.

Figure 12-6 lists the ONCONFIG parameters that you use to configure the virtual portion of shared memory.

Figure 12-6
Configuring the Virtual Portion of Shared Memory

ONCONFIG Parameter	Purpose	Reference
LOGFILES	Specifies the number of logical-log files that the database server is to create during disk initialization	page 33-42
SHMVIRTSIZE	Specifies the initial size of the virtual portion of shared memory	page 33-83
STACKSIZE	Specifies the stack size for the database server user threads	page 33-85

UNIX

Setting Parameters for Shared-Memory Performance Options with ON-Monitor

To set the configuration parameters for the shared-memory performance options using ON-Monitor, select **Parameters→perFormance**. Figure 12-7 shows the perFormance screen. The shaded entries set the configuration parameters for the shared-memory performance options.

```
PERFORMANCE: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                    PERFORMANCE TUNING PARAMETERS

 Multiprocessor Machine      [ ]     LRU Max Dirty             [ 60]
   Num Procs to Affinity     [  0]   LRU Min Dirty             [ 50]
   Proc num to start with    [  0]   Checkpoint Interval      [ 300]
                                     Num of Read Ahead Pages   [ 50]
 CPU VPs                     [  1]   Read Ahead Threshold      [ 40]
 AIO VPs                     [   ]
 Single CPU VP                 [N]   NETTYPE settings:
 Use OS Time                   [N]     Protocol Threads Users VP-class
 Disable Priority Aging        [N]     [soctcp] [  1]   [ 30] [NET]
 Off-Line Recovery Threads   [ 10]     [       ] [   ]   [   ] [   ]
 On-Line Recovery Threads    [  1]     [       ] [   ]   [   ] [   ]
 Num of LRUS queues          [  8]     [       ] [   ]   [   ] [   ]
```

Figure 12-7
ON-Monitor perFormance Screen

Figure 12-8 shows only the perFormance screen entries for setting the shared-memory performance options. For each entry, it shows the name of the associated parameter in the ONCONFIG file within brackets ([ ]).

```
PERFORMANCE: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                    PERFORMANCE TUNING PARAMETERS

                                     Num of LRUS queues           [LRUS]
                                     LRU Max Dirty       [LRU_MAX_DIRTY]
                                     LRU Min Dirty       [LRU_MIN_DIRTY]
                                     Checkpoint Interval     [CKPTINTVL]
                                     Num of Read Ahead Pages  [RA_PAGES]
                                     Read Ahead Threshold [RA_THRESHOLD]
```

Figure 12-8
ONCONFIG Parameters for the Shared-Memory Performance Options

For more information on the ONCONFIG parameters that set shared-memory performance options, refer to Figure 12-9 on page 12-13.

Setting Parameters for Shared-Memory Performance Options with a Text Editor

You can use a text editor to set ONCONFIG parameters at any time. To change one of the configuration parameters that set shared-memory performance options, use the text editor to locate the parameter in the file, enter the new value or values, and rewrite the file to disk. The changes that you make do not take effect until you reinitialize shared memory.

Figure 12-9 lists the ONCONFIG parameters that set shared-memory performance options. The page references in the third column refer to descriptions of the parameters in Chapter 33, "Configuration Parameters."

Figure 12-9
Setting Shared-Memory Performance Options

ONCONFIG Parameter	Purpose	Reference
CKPTINTVL	Specifies the maximum number of seconds that can elapse before the database server checks if a checkpoint is needed	page 33-13
LRU_MAX_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to start	page 33-46
LRU_MIN_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to stop	page 33-46
LRUS	Specifies the number of LRU queues for the shared-memory buffer pool	page 33-45
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	page 33-72
RA_THRESHOLD	Specifies the number of unprocessed memory pages that, after they are read, cause the database server to read ahead on disk	page 33-73

Reinitializing Shared Memory

The database server reinitializes shared memory when you take the database server from off-line mode to quiescent mode or when you take it from off-line mode directly to on-line mode. To reinitialize shared memory, first bring the database server off-line. After the database server is off-line, you need to bring it to quiescent mode or on-line mode to reinitialize shared memory. For information on how to take the database server from on-line mode to off-line, refer to Chapter 7, "Managing Database Server Operating Modes."

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 on-line mode.
- Change the RESIDENT parameter in the ONCONFIG file to turn shared-memory residency on or off for the next time that you initialize the database server shared memory.

For a description of the resident portion of shared memory, refer to "The Resident Portion of Shared Memory" on page 11-18.

Turning Residency On or Off in On-Line Mode

You can turn residency on or off while the database server is in on-line mode by using the **onmode** utility. You must be **root** or user **informix** to do this.

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 initialize shared memory.

Turning Residency On or Off for the Next Time You Reinitialize Shared Memory

You can use a text editor to turn residency on or off for the next time that you reinitialize shared memory. 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 reinitialize shared memory.

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.

Forcing a Checkpoint

You might want to force a checkpoint for any of the following reasons, as well as others:

- You have just finished building a large index. If the database server terminates before the next checkpoint, the index build will restart the next time you initialize the database server.
- You are about to attempt a system operation that might interrupt the database server. If it has been a long time since a checkpoint occurred, fast recovery could take longer than you want.
- Foreground writes are taking more resources than you want. You can manually force a checkpoint to bring this down to zero for a while.

You can force a checkpoint by executing the following command from the command line:

```
% onmode -c
```

UNIX

You can also use ON-Monitor to force a checkpoint by selecting the Force-Ckpt option from the main menu. ♦

Chapter

13

Where Is Data Stored?

Overview of Data Storage 13-3

What Are the Physical Units of Storage? 13-5
What Is a Chunk? 13-5
Limits on Chunk Size and Number 13-5
Unbuffered or Buffered Disk Access 13-6
What Is an Offset? 13-9
What Is a Page?. 13-10
What are Blobpages? 13-11
How Big Should Blobpage Be? 13-12
What Is an Extent?. 13-13
What Are Disabling I/O Errors? 13-15

What Are the Logical Units of Storage? 13-16
What Is a Dbspace? 13-16
How Can You Control Where Data Is Stored? 13-17
How Does Table Fragmentation Affect Data Storage? 13-19
What Is the Root Dbspace? 13-19
What Is a Temporary Dbspace? 13-20
What Are the Advantages of Using Temporary Dbspaces? 13-20
What Is a Blobspace? 13-21
What Is a Database? 13-22
What Is a Table? 13-23
What Is a Temporary Table? 13-24
Where Are Temporary Tables Stored?. 13-26
What Is a Tblspace? 13-27
What Is Extent Interleaving? 13-29

How Much Disk Space Do You Need to Store Your Data? 13-30
Calculate the Size of the Root Dbspace 13-30
Physical and Logical Logs 13-30
Temporary Tables 13-31
Critical Data 13-31
ON-Archive Catalog Data 13-32
Control Information (Reserved Pages) 13-32
Complete the Root Dbspace Calculation 13-32
Estimate Space That Databases Require 13-33

Disk-Layout Guidelines 13-33
Dbspace and Chunk Guidelines 13-34
Strive to Associate Partitions with Chunks 13-34
Mirror Critical Data Dbspaces 13-34
Spread Your Temporary Storage Space Across Multiple Disks 13-35
Move the Logical and Physical Logs from the Root Dbspace 13-35
Consider Account Backup-and-Restore Performance 13-36
Table-Location Guidelines 13-36
Isolate High-Use Tables 13-37
Consider Mirroring 13-38
Group Your Tables with Backup and Restore in Mind 13-38
Place High-Use Tables on Middle Partition of Disk 13-39
Optimize Table-Extent Sizes 13-40

Sample Disk Layouts 13-40
Sample Layout When Performance Is Highest Priority 13-42
Sample Layout When Availability Is Highest Priority 13-44

What Is a Logical-Volume Manager? 13-46

This chapter defines terms and explains the concepts that you must understand to perform the tasks described in Chapter 14, "Managing Disk Space." 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

Important: *The examples in this chapter use command-line utilities to illustrate how to create disk spaces for the database server. You can also use INFORMIX-Enterprise Command Center (IECC), which is a graphical interface, to accomplish the same tasks more easily. For more information managing disk space with IECC, see the "Informix Enterprise Command Center User Guide."*

The release-notes file contains supplementary information on the maximum values related to the storage units discussed in this chapter. For information on how to access this file, see "Documentation Notes, Release Notes, Machine Notes" on page 15 of the Introduction.

Overview of Data Storage

The database server can use regular operating-system files to store data. On operating systems that support raw disks, the database sever can also use raw disk space to store data. On UNIX platforms, Informix recommends that you use raw disks to store data whenever performance or data consistency is important. On Windows NT platforms, Informix recommends using unbuffered NTFS files to store data for ease of administration.

The database server uses the following physical units to manage disk space:

- Chunk
- Page
- Blobpage
- Extent

Overlying the physical units of storage space, the database server supports the following logical units associated with database management:

- Dbspace
- Blobspace
- Extspace
- Database
- Table
- Tblspace

The database server maintains the following additional disk-space storage structures to ensure physical and logical consistency of data:

- Logical log
- Physical log
- Reserved pages

Because these additional disk-space structures are not permanent storage units, they are not described in this chapter. For information about the logical log, see Chapter 18, "What Is the Logical Log?" For information about the physical log, see Chapter 20, "What Is Physical Logging?" For information about reserved pages, see "Reserved Pages" on page 38-6.

The following sections describe the various data-storage units that the database server supports and the relationships between those units.

What Are the Physical 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—chunks, extents, pages, and blobpages—has a fixed or assigned size.

What Is a Chunk?

The *chunk* is the largest unit of physical disk dedicated to database server data storage.

The database server administrator typically adds a chunk to a dbspace when that dbspace approaches full capacity.

Chunks provide administrators with a conveniently large unit for allocating disk space. The database server also uses chunks for mirroring. A *primary chunk* is a chunk from which the database server copies data to a *mirrored chunk*. If the primary chunk fails, the database server brings the mirrored chunk on-line automatically. For more information on mirroring, see Chapter 23, “What Is Mirroring?”

Some operating systems use the concept of a logical volume, and others use a logical unit. Each of these terms represents the smallest unit of physical disk that you can assign. A database server chunk is the same as a logical volume or a logical unit.

Limits on Chunk Size and Number

On most platforms, the maximum size of a chunk is 2 gigabytes, but on some platforms the maximum chunk size is 4 gigabytes.

Unbuffered or Buffered Disk Access

This section describes the advantages and disadvantages of the two methods of allocating disk space: using files that are buffered through the operating system, also referred to as *cooked* files, and using unbuffered disk access. Depending on the operating system, unbuffered disk access can be through unbuffered files, through a raw disk device, or character-special files. As a general guideline, you experience better performance and increased reliability when you use unbuffered file access.

UNIX

Disk Access on UNIX

On UNIX, the raw disk interface that character-special files provide yields significant performance advantages. I/O to raw disk bypasses the buffering operations that the operating system performs on regular (cooked) files.

WIN NT

Disk Access on Windows NT

In the Windows NT environment, the database server uses unbuffered I/O to access both *raw disk* space and NTFS files. Consequently, I/O to raw disk space in the Windows NT environment does not yield a significant improvement in performance compared to I/O to NTFS files. For this reason, Informix recommends that you use regular NTFS files for chunks.

If you use files for disk space on Windows NT, the files must be NTFS files.

UNIX

What Is Raw Disk Space on UNIX?

UNIX uses the concept of a *device* to describe peripherals such as magnetic disks and tapes, terminals, and communication lines. One type of device is a *block device*, such as a hard disk or a tape. A block device can be configured with an interface that provides buffering or with a raw interface that leaves the buffering to the application. When you configure a block device with a raw interface, the device is called a *raw device*, and the storage space that the device provides is called *raw disk space*. Space in a chunk of raw disk space is physically contiguous.

A raw interface is also referred to as a *character-special device*. 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.

WIN NT

What is Raw Disk Space on Windows NT?

In the Windows NT environment, *raw disk space* can be either a physical drive without a drive letter or a disk partition that has been assigned a drive letter using the Windows NT Disk Administrator. The space can either be formatted or unformatted but if it contains data, the data will be overwritten after the space has been allocated to the database server.

What Is a Cooked File?

A cooked file is a regular file that the operating system manages. Although the database server manages the contents of cooked files, the operating system manages all I/O to cooked files. Unlike raw disk space, the logically contiguous blocks of a cooked file might not be physically contiguous.

Even though a cooked file is a regular file, the database server manages the internal arrangement of data within the file. Never edit the contents of a cooked file that the database server manages. To do so puts the integrity of your data at risk.

How Does the Database Server Manage Data Differently When It Is Stored in a Cooked File Instead of a Raw Disk Device?

When the operating system reads from a cooked file, it reads the data from disk into an internal buffer pool. Later, a second copy operation copies it from the operating system to the location requested by the application. Therefore. when two users both read the same file, the data is read from disk only once but copied from the operating-system buffer twice.

By contrast, when the operating system reads data from an unbuffered file or a raw disk device, it bypasses the operating-system buffer pool and copies the data directly to the location requested by the application. The database server requests that the data be placed in shared memory, making it immediately available to all database server virtual processors and running threads with no further copying.

Why Use Unbuffered Disk Access?

A raw device or unbuffered file can directly transfer data between shared memory and the disk using direct memory access (DMA), which results in better performance by orders of magnitude.

When you use a raw device or unbuffered file to store your data, the database server guarantees that committed data is stored on disk. (The next section explains why no such guarantee can be made when you use cooked files to store your data.)

When you decide to allocate raw disk space to store your data, you must take the following steps:

1. Create and install a raw device.
2. Change the ownership and permissions of the device.

These steps are described in detail in "Allocating Raw Disk Space on UNIX" on page 14-8.

Why Use a Cooked File?

You can more easily allocate cooked files than raw disk space. To allocate raw space, you must have a disk partition available that is dedicated to raw space. To allocate a cooked file, you need only create the file on any existing partition. However, you sacrifice reliability and might experience diminished performance when you store the database server data in cooked files.

The buffering mechanism that most operating systems provide can become a performance bottleneck. If you must use cooked UNIX files, store the least frequently accessed data in those files. Store the files in a file system located near the center cylinders of the disk device or in a file system with minimal activity.

In a learning environment, where reliability and performance are not critical, cooked files can be convenient.

When performance is not a consideration, you can also use cooked files for static data (which seldom, if ever, changes). Such data is less vulnerable to the problems associated with UNIX buffering in the event of a system failure.

When a chunk consists of cooked disk space, the name of the chunk is the complete pathname of the file. Because the chunk of cooked disk space is an operating-system file, space in the chunk might not be physically contiguous.

Warning: *Cooked files are less reliable than raw disk space because the operating system manages I/O for a cooked file. A write to a cooked file can result in data being written to a memory buffer in the operating-system file manager instead of being written immediately to disk. As a consequence, the database server cannot guarantee that the committed data actually reaches the disk. Database server recovery depends on the guarantee that data written to disk is actually on disk. In the event of system failure, if the data is not present on disk, the database server automatic-recovery mechanism might not be able to execute properly. The end result would be inconsistent data.*

When you decide to allocate cooked space to store your data, you must take the following steps:

1. Create a cooked file.
2. Change the ownership and permissions.

These steps are described in detail in "Allocating a File for Disk Space on UNIX" on page 14-7.

What Is an Offset?

Although Informix recommends that you use an entire disk partition when you allocate a chunk on a raw disk device (see "Strive to Associate Partitions with Chunks" on page 13-34 for more information), you can subdivide partitions or cooked files into smaller chunks using *offsets*.

An offset allows you to indicate the number of kilobytes into a device or cooked file that are needed to reach a given chunk. For example, suppose that you create a 1,000 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 a dbspace or blobspace, add a chunk to a dbspace or blobspace, or drop a chunk from a dbspace or blobspace. The maximum offset that you can specify is 2 gigabytes. This offset is equal to the maximum chunk size, also 2 gigabytes.

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 14-8 explains when and how to specify an offset.

What Is a Page?

A *page* is the physical unit of disk storage that the database server uses to read from and write to Informix databases. The size of a page varies from computer to computer. A page typically holds either 2 or 4 kilobytes. Because your hardware determines the size of your page, you cannot alter this value. Figure 13-1 illustrates the concept of a page, represented by a darkened sector of a disk platter.

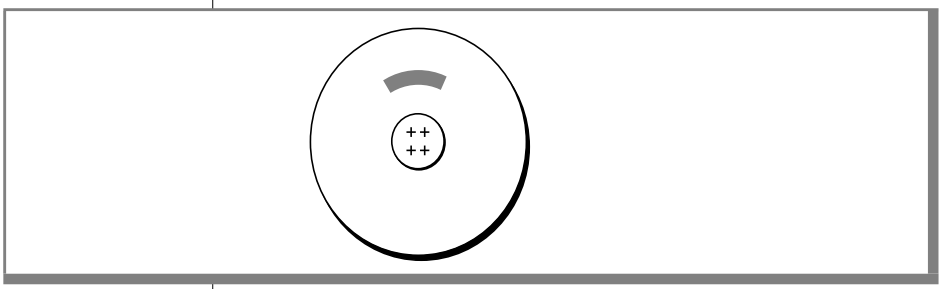


Figure 13-1
A Page on Disk

A chunk contains a certain number of pages, as illustrated in Figure 13-2. A page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.

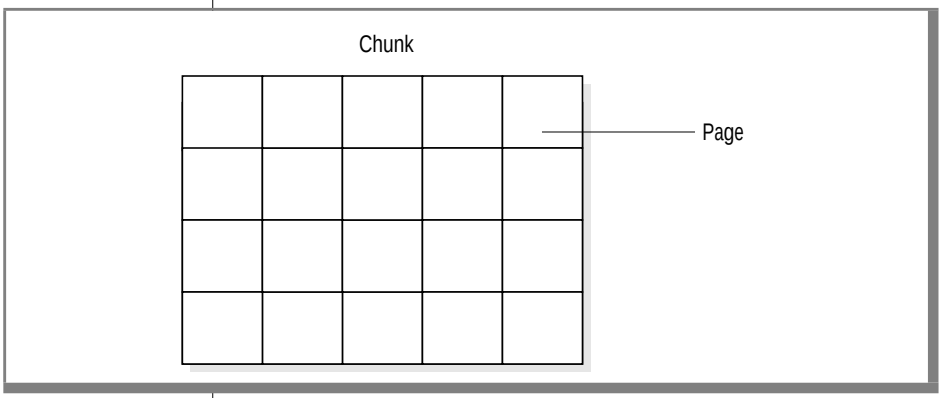


Figure 13-2
A Chunk, Logically Separated into a Series of Pages

For information on how the database server structures data within a page, see Chapter 38, "Disk Structures and Storage."

What are Blobpages?

A blobpage is the unit of disk-space allocation that the database server uses to store TEXT or BYTE data within a blobspace. You specify blobpage as a multiple of the database server page size. Because the database server allocates blobpages as contiguous spaces, it is more efficient to store TEXT or BYTE data in blobpages that are as close to the size of the data as possible. Figure 13-3 illustrates the concept of a blobpage, represented as a multiple (three) of a data page.

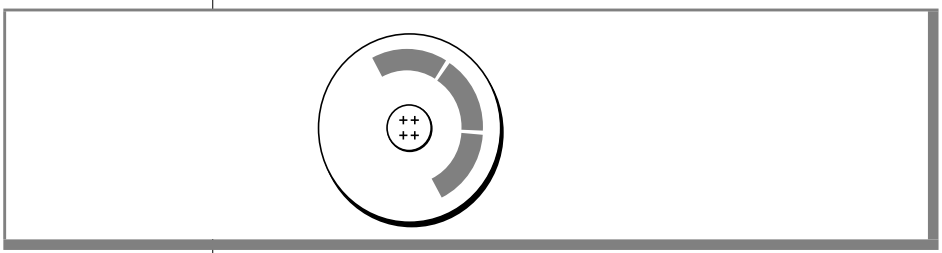


Figure 13-3
A Blobpage on Disk

Just as with pages in a chunk, a certain number of blobpages compose a chunk in a blobspace, as illustrated in Figure 13-4. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

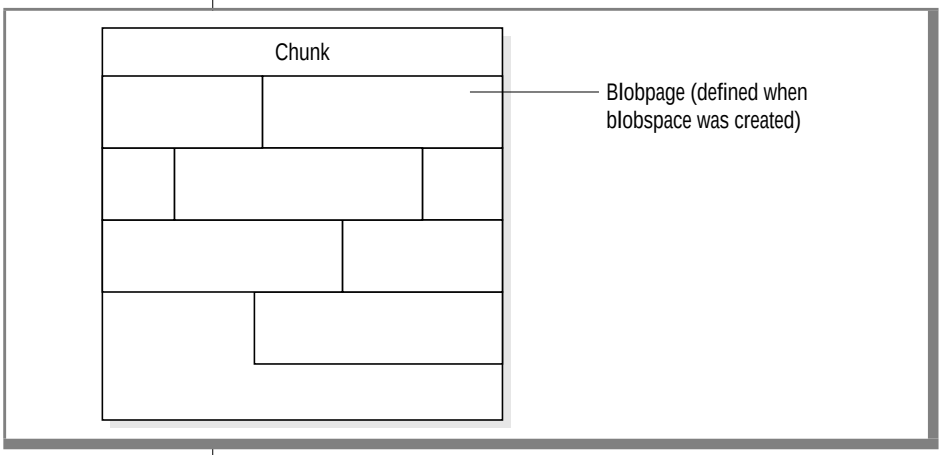


Figure 13-4
A Chunk in a Blobspace, Logically Separated into a Series of Blobpages

How Big Should Blobpage Be?

When you create a blobspace, try to create a blobpage size that approximates the size of the most frequently occurring TEXT or BYTE data that the blobspace holds. For example, if you are storing 160 TEXT or BYTE objects, and you expect 120 TEXT or BYTE objects to be 12 kilobytes and 40 TEXT or BYTE objects to be 16 kilobytes, a 12-kilobyte blobpage size stores the objects most efficiently. This configuration allows the majority (120) of the objects to be stored in a single blobpage, while the other 40 objects require two blobpages each. Even though 8 kilobytes are wasted in the second blobpage (8 kilobytes wasted space per page * 40 pages = 320 kilobytes total wasted space), this arrangement does not waste as much space as using a blobpage size of 16 kilobytes (4 kilobytes wasted space per page * 120 pages = 480 kilobytes total wasted space).

In some circumstances, you might want to use the larger, 16-kilobyte blobpage size. If speed and reducing the number of locks are primary concerns, use a 16-kilobyte blobpage so that every TEXT or BYTE object can be stored on a single blobpage. You also might want to use a 16-kilobyte blobpage size if those blobpages are accessed more frequently than the 12-kilobyte blobpages.

To continue the example, assume that your database server page size is 2 kilobytes. If you decide on a 12-kilobyte blobpage size, specify the blobpage size parameter as 6 (pages). If your database server page size is 4 kilobytes, specify the blobpage size parameter as 3 (pages). In general, divide the size of the TEXT or BYTE object (rounded up to the nearest kilobyte) by the page size to determine the blobpage size parameter.

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 "What Is a Table?" on page 13-23.

What Is an Extent?

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 13-5 illustrates the concept of an extent.

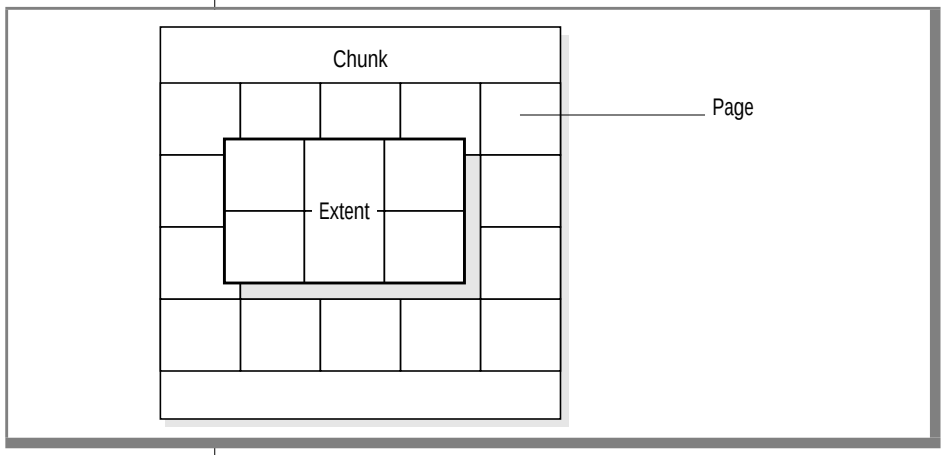


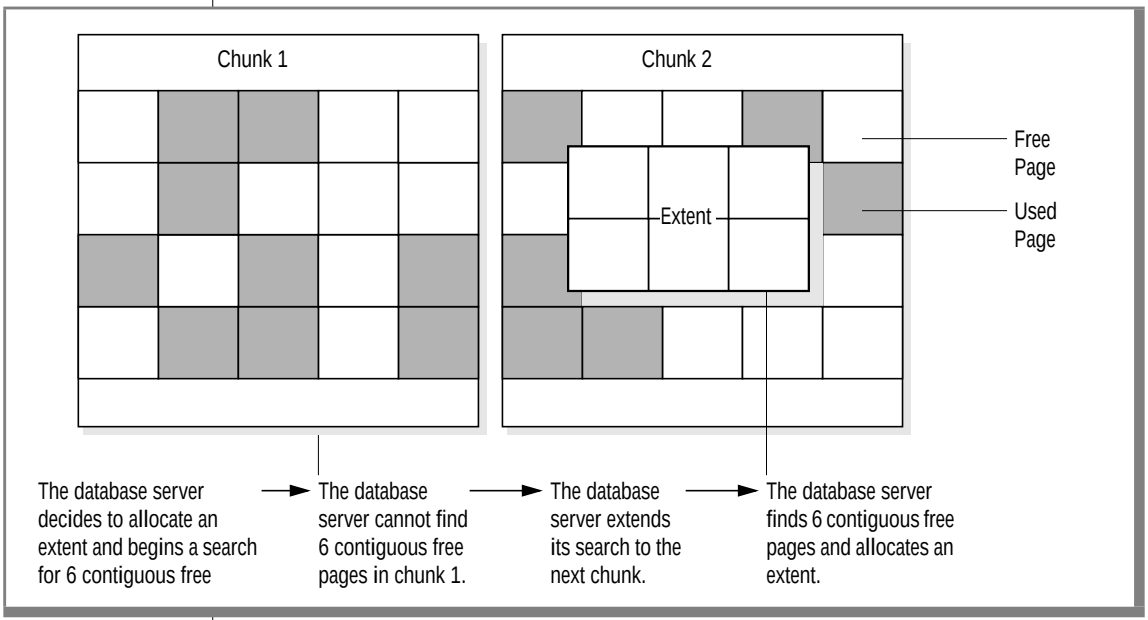
Figure 13-5
An Extent That Consists of Six Contiguous Pages on a Raw Disk Device

An extent consists of a collection of contiguous pages that store data for a given table. (See "What Is a Table?" on page 13-23.) 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. Specify the initial-extent size and next-extent size using the CREATE TABLE and ALTER TABLE statements. For more information, see the *Informix Guide to SQL: Syntax*.

Figure 13-6 illustrates the following key concepts concerning extent allocation:

- 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 (six pages in this case), it searches the next chunk in the dbspace for contiguous space.

Figure 13-6
Process of Extent Allocation



What Are 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 off-line.

Second, the error must occur when the database server attempts, but fails, 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 off-line and then on-line 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.

What Are the Logical Units of Storage?

The logical units of the database server storage fall into the following categories:

- Units of logical storage that function as accounting entities, including:
  - Dbspaces
  - Blobspace
  - Tblspaces
- Units of logical storage that are dictated by relational database design, including:
  - Databases
  - Tables

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 "What Is a Table?" on page 13-23.)

The following sections describe these logical storage units.

What Is a Dbspace?

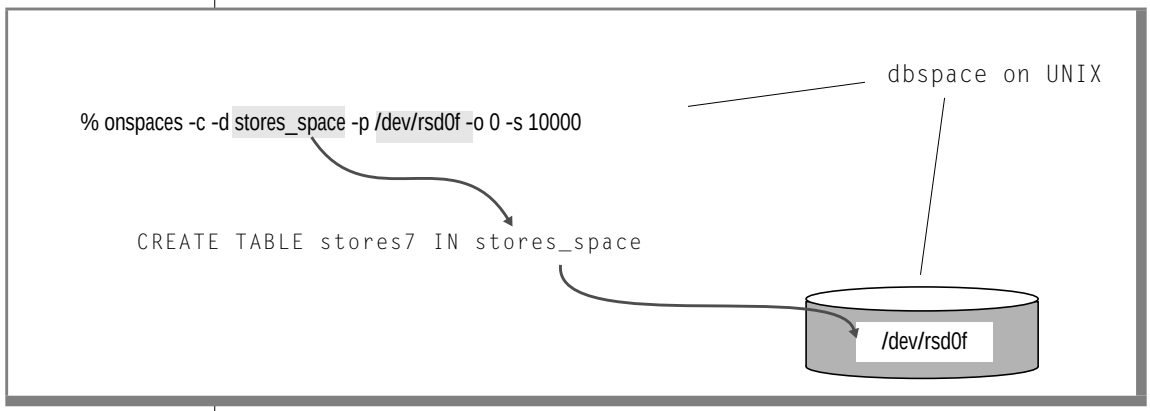
A key responsibility of the database server administrator is to control where the database server stores data. By storing high-use access tables or critical dbspaces (root dbspace, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical data on separate 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.

These strategies require the ability to control the location of data. The logical storage unit that provides this ability is the *dbspace*. The dbspace provides the critical link between the logical and physical units of storage. It allows you to associate physical units (such as chunks) with logical units (such as tables).

How Can You Control Where Data Is Stored?

As Figure 13-7 shows, you control the placement of databases or tables (see "What Is a Table?" on page 13-23) using the IN *dbspace* option of the CREATE DATABASE or CREATE TABLE statements.

Figure 13-7
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" on page 14-11.

A dbspace includes one or more chunks, as Figure 13-8 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 the Database Server for Disabling I/O Errors" on page 29-52.) When a dbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

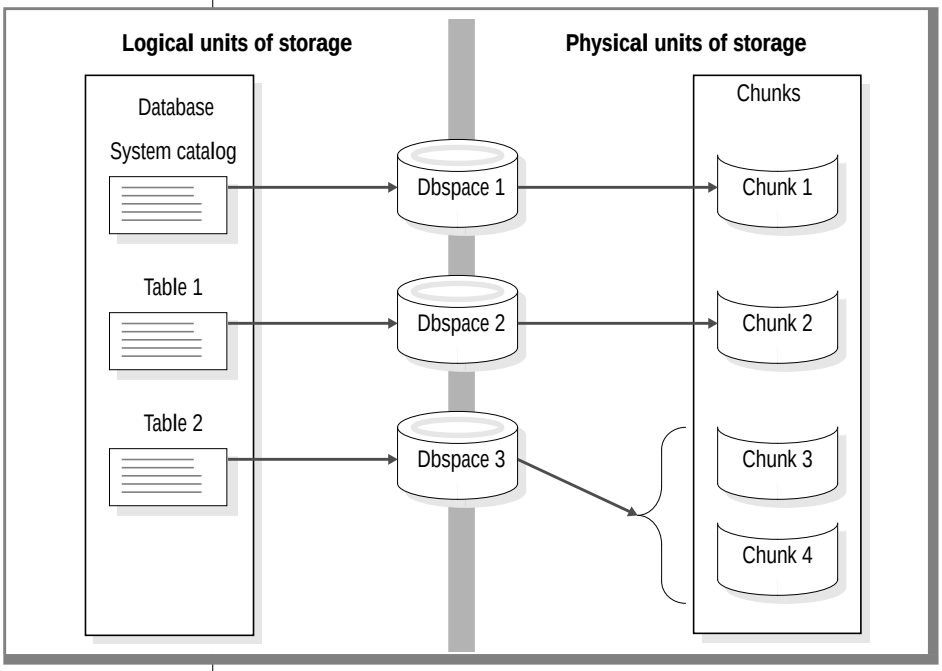


Figure 13-8
Dbspaces That Link Logical and Physical Units of Storage

The database server uses the dbspace to store databases and tables. (See "What Is a Table?" on page 13-23.)

You can mirror every chunk in a mirrored dbspace. As soon as the database server allocates a mirrored chunk, it flags all space in that mirrored chunk as full. See "Monitoring Disk Usage" on page 29-53.

You can use ON-Monitor (on UNIX), **onspaces**, or IECC to perform any of the following tasks related to dbspace management:

- Creating a dbspace (page 14-11)
- Adding a chunk to a dbspace (page 14-14)
- Dropping a dbspace or blobspace (page 14-19)

How Does Table Fragmentation Affect 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*. For more information about fragmentation, see "What Is Fragmentation?" on page 15-3.

What Is the 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 (see "What Is a Table?" on page 13-23) that describe and track all other dbspaces, blobspaces, chunks, databases, and tblspaces. (For more information on these topics, see Chapter 38, "Disk Structures and Storage.") The initial chunk of the root dbspace and its mirror are the only chunks created during disk-space initialization. You can add other chunks to the root dbspace after disk-space initialization.

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

The root dbspace is the default location for all temporary tables created implicitly by the database server to perform requested data management. The root dbspace is also the default dbspace location for any database created with the CREATE DATABASE statement.

"Calculate the Size of the Root Dbspace" on page 13-30 explains how much space to allocate for the root dbspace. You can also add extra chunks to the root dbspace after you initialize database server disk space.

What Is a Temporary Dbspace?

A temporary dbspace is a dbspace reserved for the exclusive use of temporary tables. (See "What Is a Temporary Table?" on page 13-24.)

The database server never drops a temporary dbspace unless it is explicitly directed to do so. A temporary dbspace is only temporary in the sense that the database server does not preserve any of the dbspace contents when the database server shuts down abnormally. Temporary dbspaces are designed exclusively for the storage of temporary tables.

Whenever you initialize the database server, all temporary dbspaces are reinitialized. The database server clears any tables that might be left over from the last time that the database server shut down.

The database server does not perform logical or physical logging for temporary dbspaces. Backup utilities do not include temporary dbspaces as part of a full-system dbspace backup. You cannot mirror a temporary dbspace.

For detailed instructions on how to create a temporary dbspace, see "Creating a Dbspace" on page 14-11.

Important: *When the database server is running as a secondary database server in a high-availability data-replication pair, it requires a temporary dbspace to store any internal temporary tables generated by read-only queries.*

What Are the Advantages of Using Temporary Dbspaces?

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 suppresses all logical logging for implicit temporary tables and explicit temporary tables created with the WITH NO LOG options that reside in a temporary dbspace. 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.

The database server does not perform any physical logging in temporary dbspaces. This practice helps performance in two ways. First, physical logging itself generates I/O. Reducing I/O always improves performance. Second, whenever the physical log becomes 75 percent full, a checkpoint occurs. Checkpoints require a brief period of inactivity to complete, which can have a negative impact on performance. When temporary tables reside in temporary dbspaces, the database server does not perform physical logging for operations on the temporary tables, thus requiring fewer checkpoints.

Using temporary dbspaces to store temporary tables also reduces the size of your dbspace backup because the database server does not backup temporary dbspaces.

What Is a Blobspace?

A blobspace is a logical storage unit composed of one or more chunks that store only TEXT and BYTE data, also called *blobs*. 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 "What Is a Table?" on page 13-23) 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 TEXT and BYTE Data in a Blobspace" on page 29-63. For instructions on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see Chapter 14, "Managing Disk Space."

For information about the structure of a blobspace, see "Structure of a Blobspace" on page 38-59.

What Is a Database?

A database is a logical storage unit that contains tables (see "What Is a Table?" on page 13-23) and indexes. Each database also contains a system catalog that tracks information about many of the elements in the database, including tables, indexes, stored procedures, and integrity constraints. Figure 13-9 on page 13-22 shows the tables contained in the **stores7** database.

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

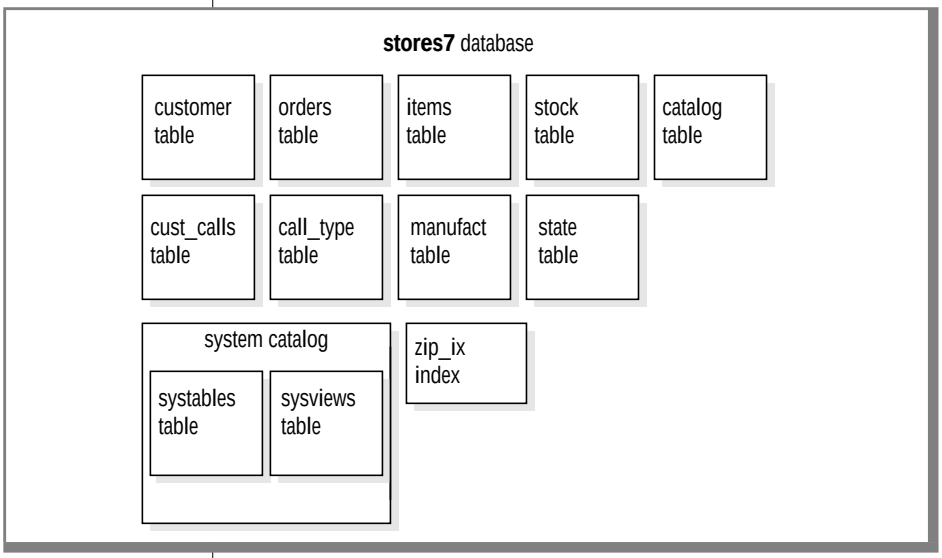


Figure 13-9
The stores7 Database

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 "Monitoring Databases" on page 29-43.

What Is a Table?

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 users create a table, the database server allocates disk space for the table in a block of pages called an extent. (See "What Is an Extent?" on page 13-13.) You can specify the size of both the first and any subsequent extents.

Users 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 the user does not specify the dbspace, the database server places the table in the dbspace where the database resides.

Users can also fragment a table over more than one dbspace. Users must define a distribution scheme for the table that specifies which table rows are located in which dbspaces.

Users can also fragment a table over more than one dbspace. Users must define a distribution scheme for the table that specifies which table rows are located in which dbspaces. For more information about distribution schemes, see the *Informix Guide to Database Design and Implementation*.

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.

A table, composed of extents, can span multiple chunks, as Figure 13-10 shows.

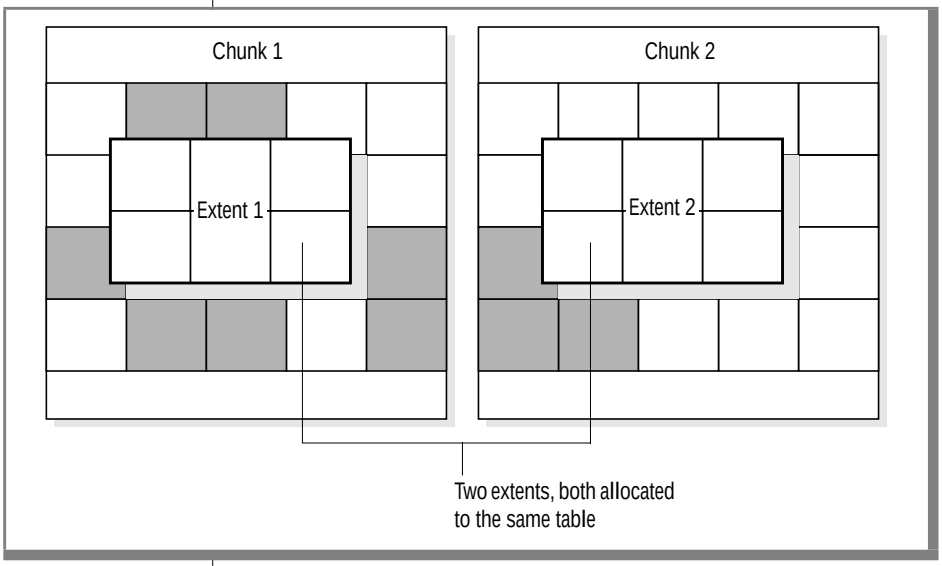


Figure 13-10
Table That Spans More than One Chunk

TEXT or BYTE data can reside either in the dbspace with the rest of the table data or in a separate blobspace. When you use the Optical Subsystem, you can also store TEXT or BYTE data in an optical storage subsystem.

For advice on where to store your tables, see "Disk-Layout Guidelines" on page 13-33 and your *Performance Guide*.

What Is a Temporary Table?

The two types of temporary tables are *explicit* temporary tables and *implicit* temporary tables.

An *explicit* temporary table is a temporary table that you create using the TEMP TABLE option of the CREATE TABLE statement or the INTO TEMP clause of the SELECT statement. For instance, the following SQL statement explicitly creates a temporary table:

```
SELECT * FROM customer INTO TEMP temp_table
```

When an application creates an explicit temporary table, it exists until the application takes one of the following actions:

- The application terminates.
- The application closes the database in which the table was created and opens a database in a different database server.
- The application closes the database in which the table was created.

  In this case, the database server drops the table only when the database uses transaction logging and the temporary table was not created with the WITH NO LOG option.

When any of these three events occurs, the database server deletes the temporary table.

An *implicit* temporary table is a temporary table that the database server creates as part of processing.

The following statements might require temporary disk space:

- Statements that include a GROUP BY or ORDER BY clause
- Statements that use aggregate functions with the UNIQUE or DISTINCT keywords
- Statements that use auto-index 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
- Statements that initiate a sort-merge join
- CREATE INDEX statements
- DECLARE statements that use the SCROLL CURSOR option

The database server deletes an implicit temporary table when the processing that initiated the creation of the table is complete.

If the database server shuts down without adequate time to clean up temporary tables, it performs temporary table cleanup as part of the next initialization. (To request shared-memory initialization without temporary table cleanup, execute **oninit** with the **-p** option.)

Where Are Temporary Tables Stored?

The dbspace in which the database server stores temporary tables depends on whether the table is an explicit or implicit table. The following sections examine both cases in detail.

Explicit Temporary Tables

When you create an explicit temporary table using the IN *dbspace* option of CREATE TEMP TABLE, the database server stores the temporary table in that dbspace.

When you do not use the IN *dbspace* option of CREATE TEMP TABLE, or when you create the explicit table with SELECT... INTO TEMP, the database server checks the **DBSPACETEMP** environment variable and the DBSPACETEMP configuration parameter. (The environment variable supersedes the configuration parameter.) When **DBSPACETEMP** is set, the database server stores the explicit temporary table in one of the dbspaces specified in the list.

The database server keeps track of the last dbspace in the list that it used to store a temporary table. When the database server receives another request for temporary storage space, it uses the next dbspace in the list. In this way, the database server spreads I/O evenly across the temporary storage space that you specify in **DBSPACETEMP**.

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 (nontemporary) 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.

Implicit Temporary Tables

The database server stores implicit temporary tables in one of the dbspaces that you specify in the **DBSPACETEMP** environment variable or the DBSPACETEMP configuration parameter. (The environment variable supersedes the configuration parameter.) When **DBSPACETEMP** is not set, the database server stores the temporary table in the root dbspace.

When the database server creates implicit temporary tables in the process of sorting, it checks the **PSORT_DBTEMP** environment variable in addition to checking the **DBSPACETEMP** environment variable and the DBSPACETEMP configuration parameter. For further information on this topic, see your *Performance Guide*.

What Is a TbIspace?

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).

The 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)
- Bit-map pages that track page use within the table extents

Figure 13-11 illustrates the tblspaces for three tables that form part of the **stores7** database. Only one table (or table fragment) exists per tblspace. Blobpages represent TEXT or BYTE data stored in a dbspace.

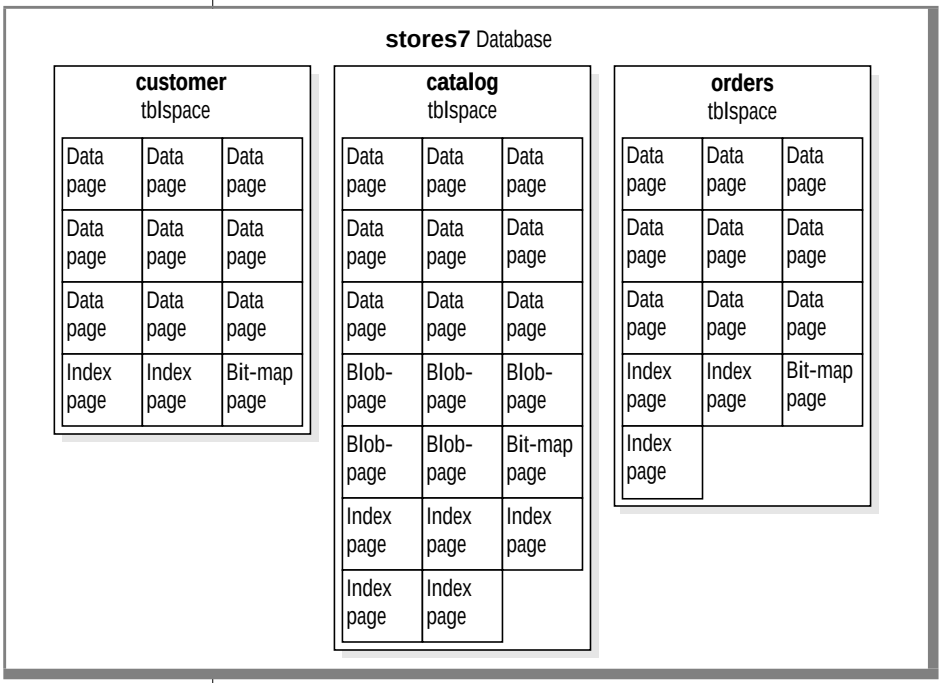
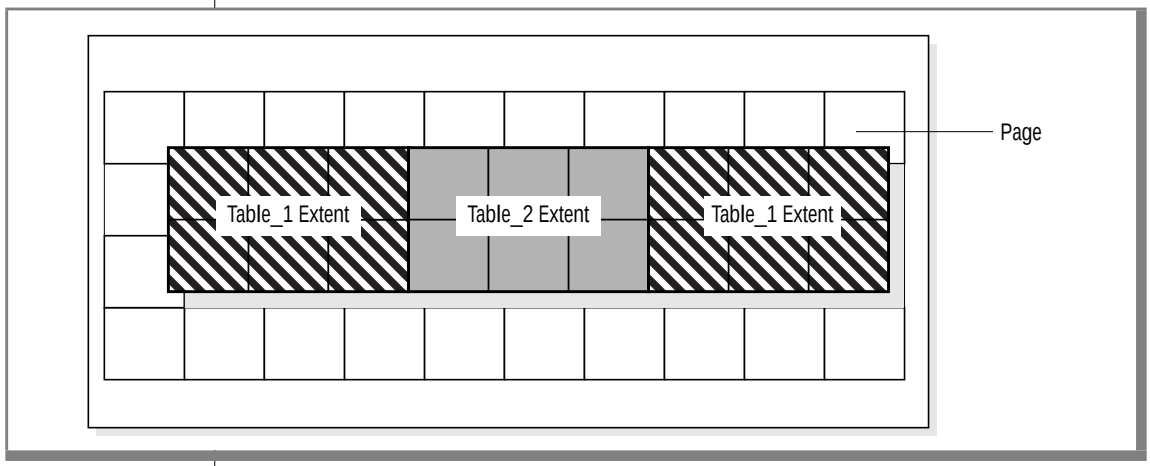


Figure 13-11
Three of the Sample Tblspaces in the stores7 Database

What Is 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 13-12 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 *Performance Guide*.

Figure 13-12
Three Extents That Belong to Two Different TbIspaces in a Single Dbspace



How Much Disk Space Do You Need to Store Your Data?

Answering the question "How much space?" is a two-step process. You must follow these steps:

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.

These steps are explained in the following sections.

Calculate the Size of the Root Dbspace

To calculate the size of the root dbspace, take the following storage structures into account:

- The physical- and logical-log files
- Temporary tables
- Data
- ON-Archive catalog data ♦
- Control information (reserved pages)

UNIX

The sections that follow discuss each of these storage structures in turn.

Physical and Logical Logs

The value stored in the ONCONFIG parameter PHYSFILE defines the size of your physical log. Advice on sizing your physical log is contained in "How Big Should the Physical Log Be?" on page 20-5.

To calculate the size of the logical-log files, multiply the value of the ONCONFIG parameter LOGSIZE by the number of logical-log files. Advice on sizing your logical log is contained in "Logical-Log Size Guidelines" on page 18-6.

Temporary Tables

Analyze end-user applications to estimate the amount of disk space that the database server might require for implicit temporary tables. "What Is a Temporary Table?" on page 13-24 contains a list of statements that require temporary space. 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 database server creates implicit temporary files when you perform a warm restore. The largest implicit temporary file 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 multiplying the value of LOGSIZE by LOGFILES. For more information on these configuration parameters, see "What Should Be the Size and Number of Logical-Log Files?" on page 18-11.

You must also analyze end-user applications to estimate the amount of disk space that the database server might require for explicit temporary tables. See "What Is a Temporary Table?" on page 13-24.

By default, the database server stores both implicit and explicit temporary tables in the root dbspace. However, if you decide not to store your temporary tables in the root dbspace, you can use the **DBSPACETEMP** environment variable and configuration parameter to specify a list of dbspaces for temporary files and tables. See "Where Are Temporary Tables Stored?" on page 13-26.

Critical Data

Next, decide if users store databases or tables in the root dbspace. If the root dbspace is the only dbspace that you intend to mirror, place all critical data there for protection. Otherwise, store databases and tables in another dbspace.

Estimate the amount of disk space, if any, that you need to allocate for tables stored in the root dbspace.

UNIX

ON-Archive Catalog Data

If you use ON-Archive to perform dbspace and logical-log backups, include space estimates for ON-Archive catalog data. For a description of the tables that compose the ON-Archive catalog, refer to your *Archive and Backup Guide*. For instructions on how to calculate the size of each table, see your *Performance Guide*.

Control Information (Reserved Pages)

The total amount of disk space required for the database server control information is 3 percent of the size of the root dbspace (sum of physical and logical log, temporary space, and data) plus 25 pages, expressed as kilobytes (or 25 times the database server page size).

Complete the Root Dbspace Calculation

Now calculate the size of the root dbspace, adding the following values for a root dbspace size:

- Physical log
- Logical log
- Disk space for temporary tables
- Disk space for data stored in the root dbspace
- Disk space for the reserved pages

UNIX

- Disk space to accommodate ON-Archive catalog data, if you use ON-Archive for dbspace and logical-log file backups ♦

You need not store the physical log, the logical log, or the temporary tables in the root dbspace. Include calculations for these entities only if you plan to continue to store them in the root dbspace.

If you plan to move the physical and logical logs, the initial configuration for the root dbspace might differ markedly from the final configuration; you can resize the root dbspace after you remove the physical and logical logs. However, the root dbspace must be large enough for the minimum size configuration during disk initialization.

Estimate 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):

1. Decide how many databases and tables you need to store. Calculate the amount of space required for each one.
2. Calculate a growth rate for each table and assign some amount of disk space to each table to accommodate growth.
3. Decide which databases and tables you want to mirror.

For instructions about calculating the size of your tables, refer to your *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 between 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.

The sections that follow discuss various strategies for meeting disk-layout goals.

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.

Strive to Associate Partitions with Chunks

When you allocate disk space (buffered or unbuffered files), you allocate it in chunks. A dbspace or a blobspace is associated with one or more chunks. You must allocate at least one chunk for the root dbspace.

Informix recommends that you format your disks so that each chunk is associated with its own disk partition. You can easily track disk-space use when you define every chunk as a separate partition (or device). You can also avoid errors caused by miscalculated offsets.

A disk that is already partitioned might require the use of offsets. For details, see "Allocating Raw Disk Space on UNIX" on page 14-8.

Mirror Critical Data Dbspaces

Mirror the critical dbspaces: the root dbspace, the dbspace that contains the physical log, and the dbspace that contains the logical-log files. You specify mirroring on a chunk-by-chunk basis. Locate the primary and the mirrored chunk on different disks. Ideally, different controllers handle the different disks. Figure 13-13 shows a primary chunk and its mirror.

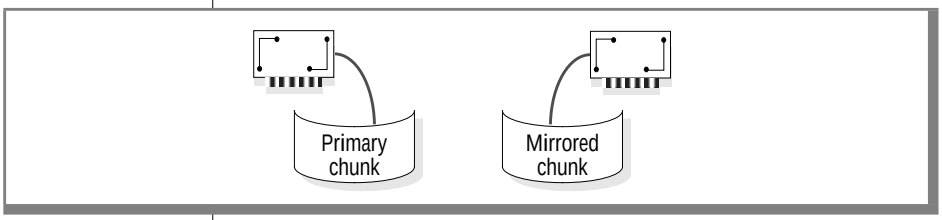


Figure 13-13
Ideal Disk Layout for Primary Chunk and Associated Mirrored Chunk

Spread Your Temporary Storage Space Across Multiple Disks

You can use the **DBSPACETEMP** environment variable and configuration parameter to store a list of dbspaces used for temporary storage. The list can include both temporary and standard dbspaces. To achieve load balancing, design the list so that your temporary disk space is spread across multiple disks. For instructions on how to set the DBSPACETEMP configuration parameter, see "DBSPACETEMP" on page 33-19.

Move the Logical and Physical Logs from the Root Dbspace

Whether or not databases use transaction logging, the logical log and physical log both contain data that the database server accesses frequently. Reserved pages are also accessed frequently; they contain internal tables that describe and track all dbspaces, blobspaces, chunks, databases, and tblspaces.

By default, the database server stores the logical and physical logs together with the reserved pages in the root dbspace. Storing the logical and physical logs together is convenient if you have a small, low-volume transaction-processing system. However, maintaining these files together in the root dbspace can become a source of contention as your database system grows.

To reduce this contention and provide better load balancing, move the logical and physical logs to separate partitions or, even better, separate disk drives. For optimum performance, consider creating two additional dbspaces: one for the physical log and one for the logical log. When you move the logs, avoid storing them in a dbspace that contains high-use tables. Instead, consider storing them in a dbspace dedicated to storing only the physical or logical log. For more advice on where to store your logs, see "Where Is the Physical Log Located?" on page 20-8 and "Where Should Logical-Log Files Be Located?" on page 18-12.

For instructions on how to change the location of the logical and physical log, see "Changing the Physical-Log Location and Size" on page 21-3 and "Moving a Logical-Log File to Another Dbspace" on page 19-8.

Consider Account Backup-and-Restore Performance

When you plan your disk layout, consider how the configuration that you choose affects your backup-and-restore procedure. This section describes two configurations that can have a significant impact on your backup-and-restore procedure.

Cluster Catalogs with the Data That They Track

When a disk that contains the system catalog of a particular database fails, the entire database remains inaccessible until you restore the system catalog. Informix recommends that you do not cluster the system catalog tables for all databases in a single dbspace but instead place the catalogs with the data that they track.

Reconsider Separating the Physical and Logical Logs

Although it makes sense from a performance perspective to separate the root dbspace from the physical and logical logs, and the two logs from one another, this configuration is the least desirable in terms of recovery.

Whenever a disk that contains critical information (the root dbspace, physical log, and logical log) fails, the database server comes off-line. In addition, the database server administrator must restore all the database server data, starting in off-line mode, from a level-0 backup before processing can continue.

When you separate the root dbspace from the physical- and logical-log files, you increase the probability that, if a disk fails, it is one that contains critical information (either the root dbspace, physical log, or logical log). For information on how to fragment to improve backup and restore characteristics, see your *Performance 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. For a discussion of how to optimize your disk layout using table fragmentation, refer to your *Performance Guide*.

Isolate High-Use Tables

You can place a table with high I/O activity on a disk device dedicated to its use and thus reduce contention for the data stored in the table. When disk drives have different performance levels, you can put the tables with the highest frequency of use on the fastest drives. Placing two high-use tables on separate disk devices reduces competition for disk access when joins are formed between the two tables or when the two tables experience frequent, simultaneous access from multiple applications.

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. Figure 13-14 illustrates this strategy by showing optimal placement of three frequently used tables.

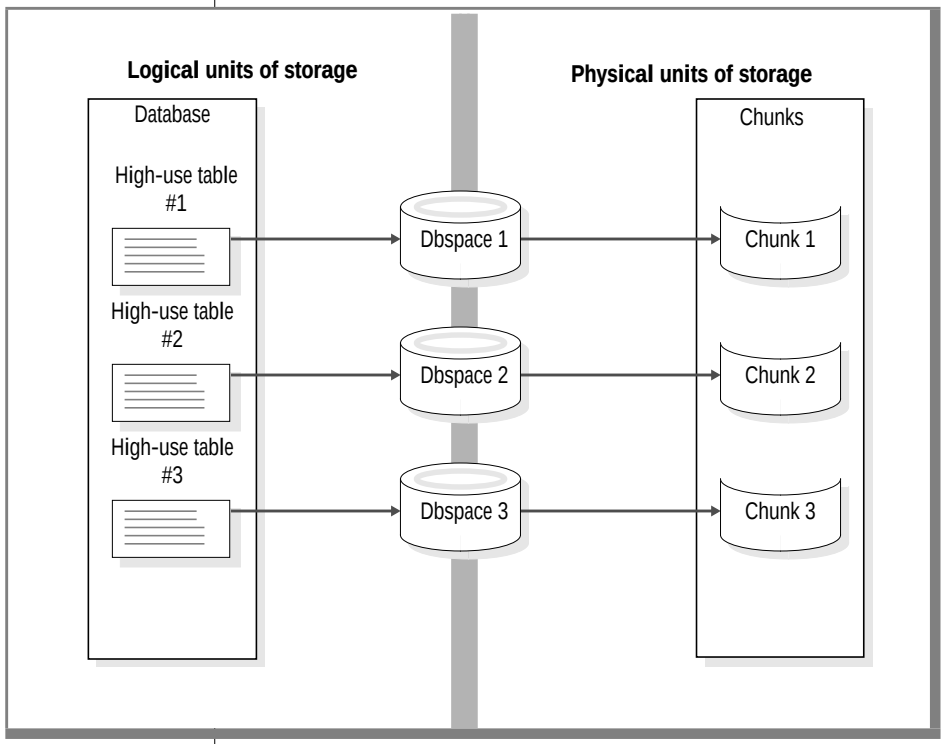


Figure 13-14
Example of High-Use Table Isolation

To take this strategy a step further, fragment a high-use table over multiple disk devices. If you choose an appropriate distribution scheme, the database server routes queries to the appropriate fragment, thereby reducing contention on any single fragment. For more information, see your *Performance Guide*.

If you have doubts whether spreading your tables across multiple disks can improve performance for your particular configuration, run the **-g iof** option of **onstat**. This option displays the level of I/O operations against each chunk. For details, see "-g Monitoring Options" on page 35-76.

Consider Mirroring

You can mirror critical tables and databases to maximize availability. You specify mirroring on a chunk-by-chunk basis. Locate the primary and mirrored chunks for critical tables on different disks. Ideally, different controllers handle the different disks.

Fragmentation gives you a higher level of control over this process. That is, you can mirror chunks that contain specific table fragments. For more information, see your *Performance Guide*.

Group Your Tables with Backup and Restore in Mind

When you decide where to place your tables, keep in mind that if a device containing a dbspace fails, all tables in that dbspace are inaccessible. However, tables in other dbspaces remain accessible. The accessibility (or inaccessibility) of dbspace might influence which tables you group together in a particular dbspace.

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. This situation might influence which dbspace you use to store critical information. If you use ON-Bar for your backup and restore tool, refer to your *Backup and Restore Guide* for more information. If you use ON-Archive or **ontape** for your backup and restore tool, refer to your *Archive and Backup Guide* for more information.

Fragmentation gives you greater granularity of backup and restore. When you fragment a table, you can still access the fragments located in the other dbspaces in the event of a dbspace failure. For more information, see your *Performance Guide*.

Place High-Use Tables on Middle Partition of Disk

To minimize disk-head movement, place the most-frequently accessed data in partitions as close to the middle of the disk as possible. See Figure 13-15. When a disk device is partitioned, the central partitions generally experience the fastest access time. Place the least-frequently used data on the outermost or innermost partitions. This overall strategy minimizes disk-head movement.

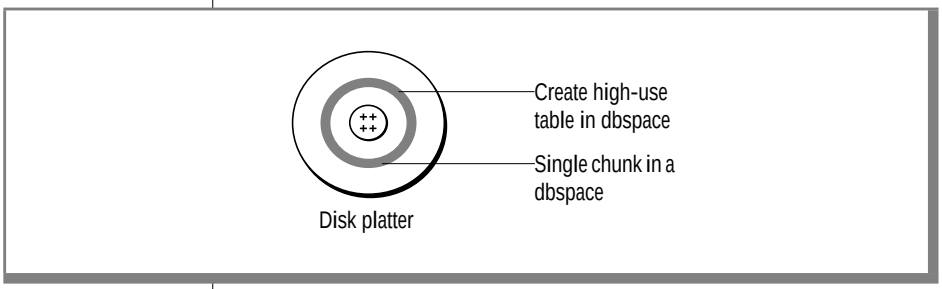


Figure 13-15
Disk Platter with High-Use Table Located on Middle Partitions

To place high-use tables on the middle partition of the disk, create a chunk using raw disk space that is composed of cylinders that reside midway between the spindle and the outer edge of the disk. Then create a dbspace with this same chunk as the initial and only chunk. When you create your high-use tables, place them in the newly created dbspace using the IN clause of the CREATE TABLE statement. For information about using raw disk space, see "What Is Raw Disk Space on UNIX?" on page 13-6 and "Disk Access on Windows NT" on page 13-6.

Optimize Table-Extent Sizes

When two or more large, growing tables share a dbspace, their new extents can become interleaved (see "What Is a Tblspace?" on page 13-27). This interleaving creates gaps between the extents of any one table. (See Figure 13-12 on page 13-29.) Performance might suffer if disk seeks must span more than one extent. Work with the table owners to optimize the table extent sizes and thus limit head movement. For advice on how to alleviate this problem, see your *Performance Guide*. You can also consider placing the tables in separate dbspaces.

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 **stores7** 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	1.5 gigabytes	No
Disk 2	2 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	1.5 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
state	50 megabytes	Low
call_type	50 megabytes	Low

Sample Layout When Performance Is Highest Priority

Figure 13-16 on page 13-43 shows a disk layout optimized for performance. This disk layout uses the following strategies to improve performance:

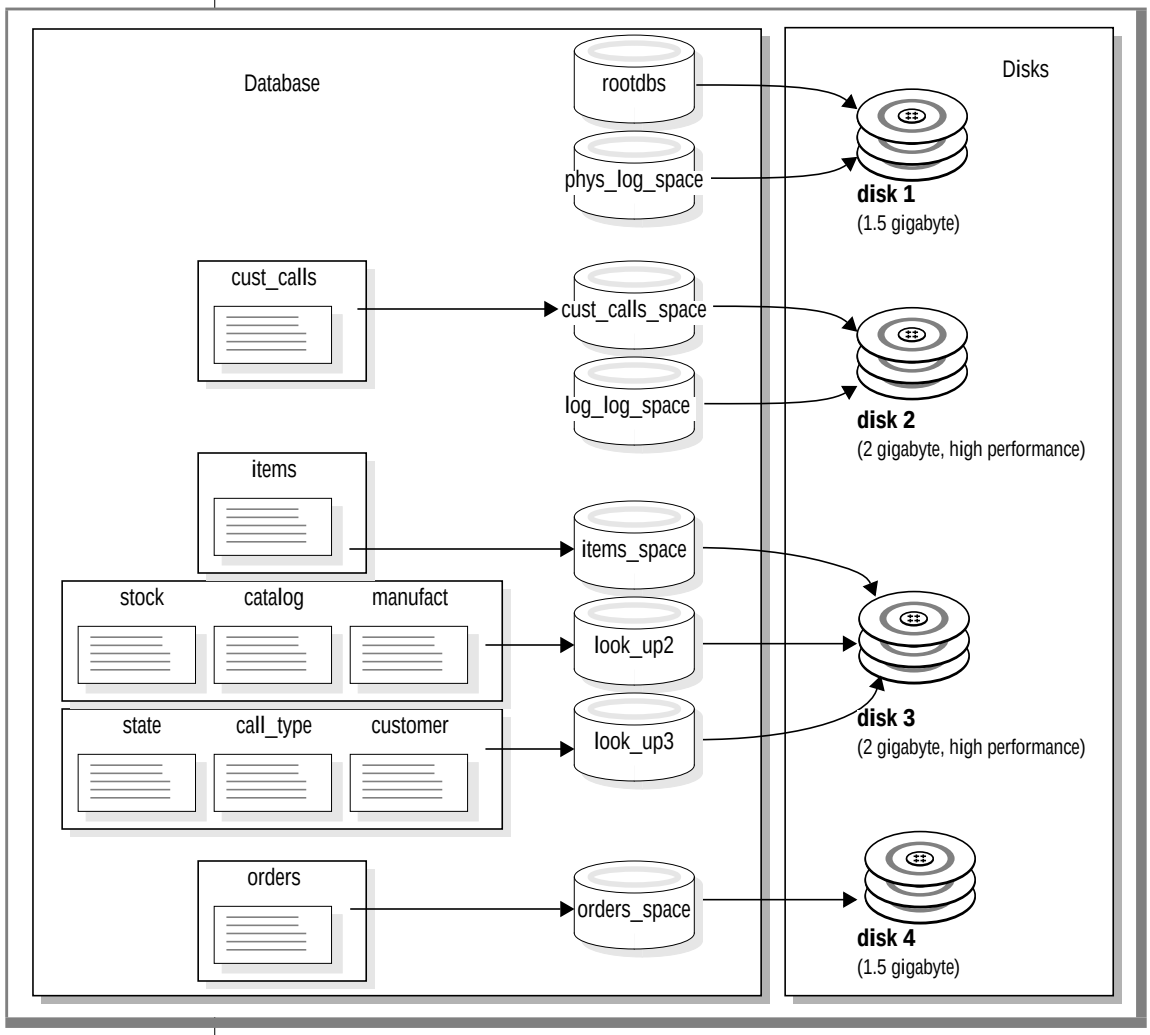
- Migration of the logical log from the rootdbs dbspace to a dbspace on a separate disk

  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 13-16
Disk Layout Optimized for Performance

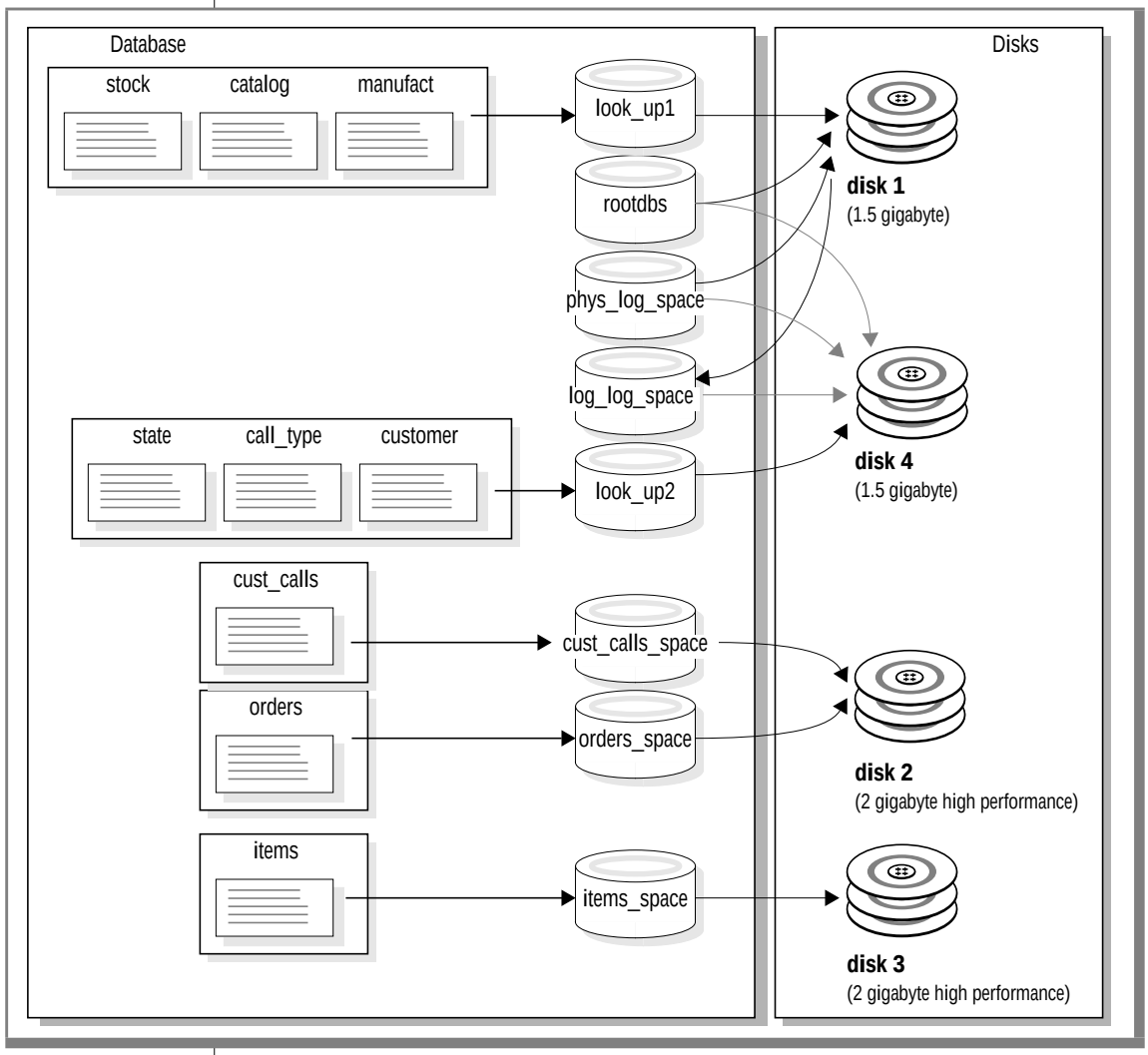


Sample Layout When Availability Is Highest Priority

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 is shown in Figure 13-16 on page 13-43. 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 13-17
Disk Layout Optimized for Availability.



What Is a 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 2 gigabytes, 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 2 gigabytes.

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. For more information, see "Place High-Use Tables on Middle Partition of Disk" on page 13-39. (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 "How Can You Control Where Data Is Stored?" on page 13-17.)

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. This strategy helps reduce contention between programs that access the same table, as explained in "Place High-Use Tables on Middle Partition of Disk" on page 13-39.

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 "What Mirroring Alternatives Exist?" on page 23-6.

Figure 13-18 illustrates the role of fragments in specifying the location of data.

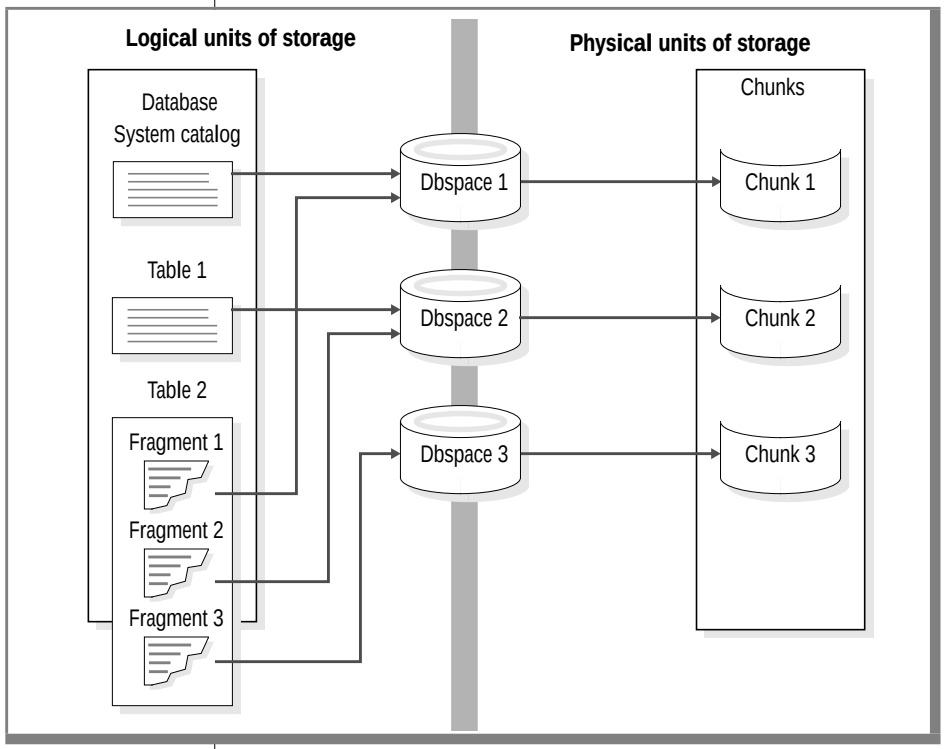


Figure 13-18
Dbspaces That Link Logical Units (Including Table Fragments) and Physical Units of Storage

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;
```

or

```
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. For more information on fragmentation schemes, refer to *Informix Guide to Database Design and Implementation*

Chapter

14

Managing Disk Space

Allocating Disk Space 14-4
Do You Need to Specify an Offset? 14-5
Specifying an Offset for the Initial Chunk of Root Dbspace 14-6
Specifying an Offset for Additional Chunks 14-6
Using Offsets to Create Multiple Chunks 14-6
Allocating a File for Disk Space on UNIX 14-7
Allocating a File for Disk Space on Windows NT 14-8
Allocating Raw Disk Space on UNIX 14-8
Creating a Link to Each Raw Device on UNIX 14-9
Allocating Raw Disk Space on Windows NT. 14-9

Initializing Disk Space. 14-10
Initializing Disk Space with oninit 14-11
Initializing Disk Space with ON-Monitor 14-11

Creating a Dbspace. 14-11
Creating a Temporary Dbspace 14-12
Creating a Dbspace with onspaces 14-13
Creating a Dbspace with ON-Monitor 14-13

Adding a Chunk to a Dbspace 14-14
Adding a Chunk 14-14
Adding a Chunk with onspaces. 14-15
Adding a Chunk with ON-Monitor 14-15

Creating a Blobspace 14-16
Determining Database Server Page Size 14-17
Creating a Blobspace with onspaces. 14-17
Creating a Blobspace with ON-Monitor 14-17

Adding a Chunk to a Blobspace 14-18

Dropping a Chunk from a Dbspace with onspaces 14-18

Dropping a Chunk from a Blobspace 14-19

Dropping a Dbspace or Blobspace 14-19
Dropping a Dbspace or Blobspace with onspaces 14-20
Dropping a Dbspace or Blobspace with ON-Monitor 14-20

Optimizing Blobspace Blobpage Size 14-21
Determining Blobspace Storage Efficiency 14-21
Blobspace Storage Statistics 14-21
Determining Blobpage Fullness with oncheck -pB 14-22
Interpreting Blobpage Average Fullness 14-23
Apply Efficiency Criteria to Output 14-24

This chapter provides the instructions that you need to manage effectively the disk spaces and data that the database server controls. It assumes you are familiar with the terms and concepts contained in Chapter 13, "Where Is Data Stored?"

This chapter covers the following topics:

- How to allocate disk space
- How to initialize disk space
- How to set configuration variables related to disk management
- How to manage chunks and dbspaces
  - Allocating, adding, and dropping chunks from dbspaces and blobspaces
  - Creating and dropping dbspaces and blobspaces
  - How to optimize blobpage size
- How to skip inaccessible fragments

Important: *The examples in this chapter use command-line utilities to illustrate how to create disk spaces for the database server. You can also use INFORMIX-Enterprise Command Center (IECC), which is a graphical interface, to accomplish the same tasks more easily. For more information on managing disk space with IECC, see the "Informix Enterprise Command Center User Guide."*

Your *Performance Guide* also contains information about managing disk space. In particular, it describes how to eliminate interleaved extents and how to reclaim space in an empty extent.

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 page 13-6
- "How Much Disk Space Do You Need to Store Your Data?" on page 13-30
- "Disk-Layout Guidelines" on page 13-33

After you allocate the necessary space, you might still need to take additional steps before the database server can begin to use the space to store data. The sections that follow contain those additional steps as well.

You need to allocate disk space before you perform these tasks:

- Initializing disk space
- Creating a dbspace or blobspace
- Adding a chunk to an existing dbspace or blobspace
- Mirroring an existing dbspace or blobspace

You can allocate either an empty file or a portion of raw disk for database server disk space.

UNIX

On UNIX, if you allocate raw disk space, Informix recommends that you 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 a Link to Each Raw Device on UNIX" on page 14-9.

Using a UNIX file and its inherent operating-system interface for database server disk space also is referred to as using *cooked space.* ♦

WIN NT

On Windows NT, Informix recommends that you allocate empty NTFS files for database server disk space. For more information on this recommendation, see "Unbuffered or Buffered Disk Access" on page 13-6. ♦

Do You Need to Specify an Offset?

When you allocate a chunk of disk space to the database server, you might want to 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

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. (For convenience, it will be referred to here as the VTOC.) The VTOC is commonly stored on the first track of the drive. A table of alternate sectors and bad-sector mappings (also called 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 on these parameters, see "ROOTOFFSET" on page 33-76 and "MIRROROFFSET" on page 33-54.

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 with one of the following utilities:

- **onspaces**
- ON-Monitor ♦

UNIX

For more information on specifying an offset for chunks of database server space, see "Creating a Dbspace" on page 14-11.

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.

UNIX

Allocating a File for Disk Space on UNIX

To allocate a file for database server disk space on UNIX, log in as user **informix** and concatenate null to the filename that the database server will use disk space. The file should have permissions set to 660 (rw-rw----). Group and owner must be set to **informix**. Figure 14-1 illustrates these steps and allocates the file **/usr/data/my_chunk** for disk space.

Figure 14-1
Preparing Cooked File Space

Step	Command	Comments
1.	`su informix`	Log in as user **informix**. (Enter the password.)
2.	`cd /usr/data`	Change directories to the directory where the cooked space will reside.
3.	`cat /dev/null > my_chunk`	Create your chunk by concatenating null to a file (in this example, a file named **my_chunk**).
4.	`chmod 660 my_chunk`	Set the permissions of the file to 660 (rw-rw----).
5.	`ls -lg my_chunk  -rw-rw----  1  informix  informix  0  Oct 12 13:43 my_chunk`	Use `ls -l` if you are using System V UNIX. Verify that both group and owner of the file are **informix**. You should see something like this line (which has wrapped around).

For information on how to create a dbspace or blobspace using the file you have allocated, refer to "Creating a Dbspace" on page 14-11 and "Creating a Blobspace" on page 14-16.

WIN NT

Allocating a File for Disk Space on Windows NT

To allocate NTFS file space for database server disk space, the first step is to create a null (zero bytes) file. Perform the following steps to allocate NTFS file space for a dbspace or blobspace, including a mirror dbspace or blobspace:

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 using 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.

Once you have allocated the file space, you can create the dbspace or blobspace as you normally would, using **onspaces**. For information on how to create a dbspace or a blobspace, refer to "Creating a Dbspace" on page 14-11 and "Creating a Blobspace" on page 14-16.

You must also follow the preceding steps prior to adding a chunk to a dbspace or blobspace.

UNIX

Allocating Raw Disk Space on UNIX

For specific instructions on how to allocate raw disk space on UNIX, see your operating-system documentation.

In general, to create raw disk space, you can either repartition your disks or unmount an existing file system. In either case, take proper precautions to back up any files before you unmount the device. (See "What Is Raw Disk Space on UNIX?" on page 13-6.)

Change the group and owner of the character-special devices to **informix**. The filename of the character-special device usually begins with the letter *r*.

Verify that the operating-system permissions on the character-special devices are `crw-rw----` .

Warning: *After you create the raw device that the database server uses for disk space, carefully heed the following warnings:*

- *Do not create file systems on the same raw device that you allocate for the database server disk space.*
- *Do not use the same raw device as swap space that you allocate for the database server disk space.*

Creating a Link to Each Raw Device on UNIX

Create a link between the character-special device name and another filename with the UNIX link command, usually **ln**.

The link enables you to replace quickly the disk where the chunk is located. The convenience becomes important if you need to restore your database server data. The restore process requires all chunks that were accessible at the time of the last dbspace backup to be accessible when you perform the restore. The link means that you can replace a failed device with another device and link the new device pathname to the same filename that you previously created for the failed device. You do not need to wait for the original device to be repaired.

Execute the UNIX command **ls -lg** (**ls -l** on System V UNIX) on your device directory to verify that both the devices and the links exist. The following example shows links to raw devices. If your operating system does not support symbolic links, hard links will work as well.

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

WIN NT

Allocating Raw Disk Space on Windows NT

To allocate raw disk space on Windows NT, specify a physical drive or a disk partition for **onspaces**, rather than a filename, when you create a dbspace or add a chunk to a dbspace. The next example adds a chunk of 5000 kilobytes of raw disk space, at an offset of 5200 kilobytes, to dbspace **dpspc3**:

```
onspaces -a dbspc3 \\.\e: -o 5200 -s 5000
```

You must be a member of the **Informix-Admin** group when you create a dbspace or add a chunk to a dbspace. The raw disk space can be formatted or unformatted disk space.

Warning: *If you allocate a formatted drive or disk partition as raw disk space, and it contains data, the database server will overwrite the data when it begins to use the disk space. You are responsible to ensure that any data on raw disk space is expendable before you allocate the disk space to the database server.*

Initializing Disk Space

Disk-space initialization uses the values stored in the configuration file to create the initial chunk of the root dbspace on disk and to initialize shared memory. When you initialize disk space, shared memory is automatically initialized for you as part of the process.

Typically, you initialize disk space just once in the life of an database server. This action occurs when you bring the database server on-line for the first time.

Warning: *When you initialize the database server disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing database server, all data in the earlier database server instance becomes inaccessible and, in effect, is destroyed.*

Only user **informix** or **root** can execute **oninit** and initialize the database server. The database server must be in off-line mode when you begin initialization. As **oninit** executes, it reads the configuration file named by the environment variable **ONCONFIG**.

You can initialize disk space using one of the following utilities:

- oninit
- ON-Monitor ♦

UNIX

Initializing Disk Space with oninit

After you configure the database server (refer to Chapter 3, "Configuring the Database Server"), you can initialize disk space by executing one of the following commands:

```
% oninit -i
```

or

```
% oninit -i -s
```

The **oninit -i** option leaves the database server in on-line mode after initiation. If you use both the **-i** and **-s** options, the database server is left in quiescent mode. For reference information on executing **oninit,** see "&Initialize Disk Space and Shared Memory" on page 35-22.

UNIX

Initializing Disk Space with ON-Monitor

To initialize the database server with ON-Monitor, select **Parameters→ Initialization**. The database server displays a series of six screens. Each screen contains a number of fields that correspond to parameters in the ONCONFIG configuration file.

Creating a Dbspace

This section explains how to create a standard dbspace (see "What Is a Dbspace?" on page 13-16) and a temporary dbspace (see "What Is a Temporary Dbspace?" on page 13-20). You can use either of the following utilities to create a dbspace:

- onspaces
- ON-Monitor ♦

UNIX

Before you create a dbspace, you must first allocate disk space as described in "Allocating Disk Space" on page 14-4.

If you are using mirroring, you can mirror the dbspace when you create it. Mirroring takes effect immediately.

UNIX

WIN NT

Specify an explicit pathname for the initial chunk of the dbspace as follows:

- If you are using raw disk on UNIX ♦

  Informix recommends that you use a linked pathname. (See "Creating a Link to Each Raw Device on UNIX" on page 14-9.)

- If you are using raw disk on Windows NT

  The pathname takes the form: `\\.\x:`, where `x` specifies the disk drive or partition. ♦

- If you are using a file for database server disk space

  The pathname is the complete path and filename.

When the initial chunk of the dbspace that you are creating is file space, 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.

You must be logged in as user **informix** or **root** to create a dbspace.

If you are creating a standard dbspace, the database server can be in on-line mode. The newly added dbspace (and its mirror, if one exists) is available immediately.

Creating a Temporary Dbspace

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.
- Reinitialize the database server.

Creating a Dbspace with onspaces

To create a dbspace with **onspaces**, use the **-c** option of **onspaces** as shown in the following example.

This example creates a 10-megabyte mirrored dbspace, **dbspce1,** using raw disk space on UNIX. An offset of 5000 kilobytes is specified for both the primary and mirrored chunks.

```
onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10000 -m /dev/raw_dev2 5000
```

The following example creates a 5000 kilobyte dbspace, **dbspc3**, with an offset of 200 kilobytes, from raw disk space (drive `e:`) on Windows NT:

```
onspaces -c -d dbspc3 \\.\e: -o 200 -s 5000
```

The following example creates a 5-megabyte temporary dbspace named **temp_space**:

```
onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5000
```

For reference information on creating a dbspace with **onspaces**, see "Create a Blobspace, Dbspace, or Temporary Dbspace" on page 35-49.

UNIX

Creating a Dbspace with ON-Monitor

To create a dbspace with ON-Monitor, follow these instructions:

1. To create a dbspace, select **Dbspaces→Create**.
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. Enter the full pathname for the initial primary chunk of the dbspace 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 dbspace, enter the mirrored-chunk full pathname, size, and optional offset in the mirrored-chunk section of the screen.

Adding a Chunk to a Dbspace

If one of your dbspaces is becoming full, you might want to add a new chunk. Before you do, however, you must first allocate disk space as described in "Allocating Disk Space" on page 14-4.

Adding a Chunk

You add a *chunk* when you need to increase the amount of disk space allocated to a dbspace or blobspace.

If you are adding a chunk to a mirrored dbspace or blobspace, you must also add a mirrored chunk.

You must specify an explicit pathname for the chunk. For more information, see "Creating a Dbspace" on page 14-11.

When you add a chunk allocated as file space, the database server verifies that the disk space is sufficient for the new chunk by creating and then removing a file of the size requested. If the size of the chunk is greater than the available space on the disk, the database server might inadvertently fill your file system in the process of verifying available disk space.

You must be logged in as user **informix** or **root** to add a chunk. You can add a chunk to a dbspace using either of the following utilities:

UNIX

- **onspaces**
- ON-Monitor ♦

You can make this change while the database server is in on-line mode. The newly added chunk (and its associated mirror, if one exists) is available immediately.

Adding a Chunk with onspaces

To add a chunk to a dbspace, use the **-a** option of **onspaces** as illustrated in the following example. This example adds a 10-megabyte mirrored chunk to **blobsp3**. An offset of 200 kilobytes for both the primary and mirrored chunk is specified. If you are not adding a mirrored chunk, you can omit the **-m** option.

```
% onspaces -a blobsp3 -p /dev/raw_dev1 -o 200 -s 10000 -m /dev/raw_dev2 200
```

The next example adds a chunk of 5000 kilobytes of raw disk space, at an offset of 5200 kilobytes, to dbspace **dpspc3**.

```
onspaces -a dbspc3 \\.\e: -o 5200 -s 5000
```

For reference information on adding a chunk to a dbspace with **onspaces**, see "Add a Chunk to a Blobspace or Dbspace" on page 35-53.

UNIX

Adding a Chunk with ON-Monitor

To add a chunk to a dbspace, follow these instructions:

1. Select the Dbspaces menu, Add_chunk 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 mirrored chunk. Enter the complete pathname 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 mirrored-chunk complete pathname, size, and optional offset in the mirror-chunk section of the screen.

Creating a Blobspace

Before you create a blobspace, you must first allocate disk space as described in "Allocating Disk Space" on page 14-4.

Specify an explicit pathname for the initial chunk of the blobspace. For more information, see "Creating a Dbspace" on page 14-11

You can mirror the blobspace when you create it if mirroring is enabled for the database server. Mirroring takes effect immediately.

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 Logical-Log Files to Free Blobpages" on page 18-22.)

When the initial chunk of the blobspace that you are creating is a file, the database server verifies that 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 blobspace is created. However, the file that the database server creates for the initial chunk is not removed. Its size represents the space left on your file system before you attempted to create the blobspace. You must remove this file to reclaim the space.

You must be logged in as user **root** or user **informix** to create a blobspace. You can create a blobspace while the database server is in on-line mode. You can use one of the following tools to create a blobspace:

- **onspaces**
- ON-Monitor ♦

UNIX

Before you create a blobspace, take time to determine what blobpage size is optimal for your environment. For instructions, see "Optimizing Blobspace Blobpage Size" on page 14-21.

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:

- To view the contents of the PAGE_PZERO reserved page, run the **oncheck -pr** utility.

UNIX

- Select the Shared-Memory option of the ON-Monitor Parameters menu. ON-Monitor displays a list of shared-memory parameters. The database server page size is the last entry on the page.
- Select the Initialize option of the ON-Monitor Parameters menu. The first entry displayed on the screen is the page size. ♦

Creating a Blobspace with onspaces

To create a blobspace with **onspaces**, use the **-c** option as illustrated in the following example. This example creates 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 mirrored chunks is specified. The blobspace is created from raw disk space on a UNIX platform.

```
% onspaces -c -b blobsp3 -g 5 -p /dev/raw_dev1 -o 200 -s 10000 -m /dev/raw_dev2 200
```

For reference information on creating a blobspace with **onspaces**, see "Create a Blobspace, Dbspace, or Temporary Dbspace" on page 35-49.

UNIX

Creating a Blobspace with ON-Monitor

To create a blobspace with ON-Monitor, follow these instructions:

1. Select the Dbspaces menu, 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 (see "Determining Database Server Page Size" on page 14-17) per blobpage in the **BLOBPage Size** field. 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 pathname 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 mirror-chunk full pathname, size, and optional offset in the mirror-chunk section of the screen. ♦

Adding a Chunk to a Blobspace

Adding a chunk to a blobspace is identical to adding a chunk to a dbspace. Both procedures are explained in "Adding a Chunk" on page 14-14.

Dropping a Chunk from a Dbspace with onspaces

To drop a chunk successfully from a dbspace with **onspaces**, all pages other than overhead pages must be freed. If any pages remain allocated to nonoverhead entities, **onspaces** returns the following error:

```
Chunk is not empty.
```

If this situation occurs, execute **oncheck -pe** to determine which database server entity still occupies space in the chunk. Remove it, and reenter the **onspaces** command.

You cannot drop the initial chunk of a dbspace. Use the **fchunk** column of **onstat -d** to determine which chunk is the initial chunk of a dbspace. For more information, see "-d Option" on page 35-72.

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
```

For reference information on dropping a chunk from a dbspace with **onspaces**, see "Drop a Chunk" on page 35-55.

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 14-18 except that the database server must be in quiescent mode. Other than this condition, you need only substitute the name of your blobspace wherever a reference to a dbspace occurs.

Dropping a Dbspace or Blobspace

Before you drop a dbspace, you must first drop all databases and tables that you previously created in the 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 are residing in the dbspace or blobspace.

You cannot drop the root dbspace.

Warning: *After you drop a dbspace or blobspace the newly freed chunks are available for reassignment to other dbspaces or blobspaces. However, before you reassign the newly freed chunks, you must perform a whole-system backup. If you do not perform this backup, and you subsequently need to perform a restore, the restore might fail because the dbspace-backup reserved pages are not up-to-date. If you are using ON-Bar for your backup and restore system, refer to your "Backup and Restore Guide." If you are using ON-Archive or the* ***ontape*** *utility as your backup and restore tool, refer to your "Archive and Backup Guide."*

If you drop a dbspace or blobspace that is mirrored, the mirror spaces are also dropped.

If you want to drop only a dbspace or blobspace mirror, turn off mirroring. (See "Ending Mirroring" on page 24-12.) This action drops the dbspace or blobspace mirrors and frees the chunks for other uses.

You must be logged in as **root** or **informix** to drop a dbspace from either ON-Monitor or **onspaces**.

Dropping a Dbspace or Blobspace with onspaces

To drop a dbspace or blobspace 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
```

For reference information on dropping a dbspace or blobspace with **onspaces**, see "Drop a Blobspace or Dbspace" on page 35-52.

UNIX

Dropping a Dbspace or Blobspace with ON-Monitor

To drop a dbspace or blobspace with ON-Monitor, follow these instructions:

1. Select the Dbspaces menu, 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.

Optimizing Blobspace Blobpage Size

Familiarize yourself with the database server approach to blobspace storage of TEXT and BYTE data before you begin this section. "Structure of a Blobspace" on page 38-59 and "Blobspace Page Types" on page 38-64 provide background information for this section. This section is not applicable if you store TEXT and BYTE data in dbspaces.

Determining Blobspace Storage Efficiency

When you are evaluating blobspace storage strategy, you can measure efficiency by two criteria:

- Blobpage fullness
- Blobpages required per TEXT or BYTE object

Blobpage fullness refers to the amount of data within each blobpage. TEXT and BYTE data stored in a blobspace cannot share blobpages. Therefore, if a single TEXT or BYTE object requires only 20 percent of a blobpage, the remaining 80 percent of the page is unavailable for use. However, you want to avoid making the blobpages too small. When several blobpages are needed to store each TEXT or BYTE object, you increase the overhead cost of storage. For example, more locks are required for updates because a lock must be acquired for each blobpage.

Blobspace Storage Statistics

To help you determine the optimal blobpage size for each blobspace, use the following database server utility commands: **oncheck -pB** and **oncheck -pe**.

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 TEXT or BYTE object stored as part of the table (or database)

The **oncheck -pe** command can provide 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 number of the database server pages used by each table to store its associated TEXT and BYTE data

Determining Blobpage Fullness with oncheck -pB

The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. These statistics provide a measure of storage efficiency for individual TEXT and BYTE objects in a database or table. If you find that the statistics for a significant number of TEXT or BYTE objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobspace.

The following example retrieves storage information for all TEXT and BYTE objects stored in the table **sriram.catalog** in the **stores7** database:

```
oncheck -pB stores7:sriram.catalog
```

Figure 14-2 shows the output of this command.

```
             BLOBSpace Report for stores7:sriram.catalog

Total pages used by table7

BLOBSpace usage:
  SpacePage  Percent Full
  NameNumberPages0-25%26-50%51-75%76-100%
---------------------------------------------------------------
  blobPIC0x3000801   x
  blobPIC0x3000822   x
       ------
   Page Size is 61443

  bspc10x2000b22            x
  bspc10x2000b62                     x
       ------
   Page Size is 20484
```

Figure 14-2 *Output of oncheck -pB*

Space Name is the name of the blobspace that contains one or more TEXT or BYTE objects stored as part of the table (or database).

Page Number is the starting address in the blobspace of a specific TEXT or BYTE object.

Pages is the number of the database server pages required to store this TEXT or BYTE object.

Percent Full is a measure of the average fullness of all the blobpages that hold this TEXT or BYTE objects.

Page Size is the size in bytes of the blobpage for this blobspace. Blobpage size is always a multiple of the database server page size. See "Determining Database Server Page Size" on page 14-17 for instructions on how to obtain the page size for your database server.

The example output indicates that four TEXT or BYTE objects are stored as part of the table **sriram.catalog**. Two objects are stored in the blobspace **blobPIC** in 6144-byte blobpages. Two more objects are stored in the blobspace **bspc1** in 2048-byte blobpages.

The summary information that appears at the top of the display, **Total pages used by table** is a simple total of the blobpages needed to store TEXT or BYTE objects. The total says nothing about the size of the blobpages used, the number of TEXT or BYTE objects stored, or the total number of bytes stored.

The efficiency information displayed under the **Percent Full** heading is imprecise, but it can alert an administrator to trends in the storage of TEXT and BYTE data.

The following sections use the output shown in Figure 14-2 to demonstrate the idea of average fullness.

Interpreting Blobpage Average Fullness

The first TEXT or BYTE object listed in Figure 14-2 on page 14-22 is stored in the blobspace **blobPIC** and requires one 6144-byte blobpage. The blobpage is 51 to 75 percent full, meaning that the size is between 0.51 * 6144 = 3133 bytes and 0.75 * 6144 = 4608. The maximum size of this TEXT or BYTE object must be less than or equal to 75 percent of 6144 bytes, or 4608 bytes.

The second object listed under blobspace **blobPIC** requires two 6144-byte blobpages for storage, or a total of 12,288 bytes. The average fullness of all allocated blobpages is 51 to 75 percent. Therefore, the minimum size of the object must be greater than 50 percent of 12,288 bytes, or 6144 bytes. The maximum size of the TEXT or BYTE object must be less than or equal to 75 percent of 12,288 bytes, or 9216 bytes. The average fullness does not mean that each page is 51 to 75 percent full. A calculation would yield 51 to 75 percent average fullness for two blobpages where the first blobpage is 100 percent full and the second blobpage is 2 to 50 percent full.

Now consider the two TEXT or BYTE objects in blobspace **bspc1**. These two objects appear to be nearly the same size. Both objects require two 2048-byte blobpages, and the average fullness for each is 76 to 100 percent. The minimum size for these TEXT or BYTE objects must be greater than 75 percent of the allocated blobpages, or 3072 bytes. The maximum size for each object is slightly less than 4096 bytes (allowing for overhead).

Apply Efficiency Criteria to Output

Looking at the efficiency information for blobspace **bspc1**, a database server administrator might decide that a better storage strategy for TEXT and BYTE data would be to double the blobpage size from 2048 bytes to 4096 bytes. (Blobpage size is always a multiple of the database server page size.) If the database server administrator made this change, the measure of page fullness would remain the same, but the number of locks needed during an update of a TEXT or BYTE object would be reduced by half.

The efficiency information for blobspace **blobPIC** reveals no obvious suggestion for improvement. The two TEXT or BYTE objects in **blobPIC** differ considerably in size, and there is no optimal storage strategy. In general, TEXT or BYTE objects of similar size can be stored more efficiently than TEXT or BYTE objects of different sizes.

Chapter

15

Overview of Table Fragmentation and PDQ

What Is Fragmentation? 15-3
Fragmentation Goals 15-5
Whose Responsibility Is Fragmentation? 15-6
Fragmentation Strategies 15-6
Fragmenting Tables 15-7
Fragmenting a Temporary Table 15-8
Fragmenting Table Indexes 15-8
Using SQL Statements to Perform Fragmentation Tasks 15-9

What Is PDQ? 15-10
High Degree of Parallelism 15-11

When Should You Use PDQ? 15-12
OLTP Applications 15-13
Processing OLTP Queries 15-13
Decision-Support Applications 15-14
Processing Decision-Support Queries 15-15

How Does the Database Server Allocate Resources with PDQ? 15-16
Parameters Used for Controlling PDQ 15-16
PDQ Priority 15-17

How Does the Database Server Use PDQ? 15-18
SQL Operations That Take Advantage of PDQ 15-18
Parallel Delete 15-18
Parallel Inserts 15-19
Parallel Index Builds 15-20
SQL Operations That Do Not Use PDQ 15-21
Update Statistics 15-21
Stored Procedures and Triggers 15-22
Correlated and Uncorrelated Subqueries 15-22
Outer Index Joins 15-23
Remote Tables 15-23

This chapter provides an overview of the table fragmentation and parallel database query (PDQ) features of your database server.

Table fragmentation allows you to store the parts of a table on different disks. Table fragmentation allows you to store large amounts of data in a single table and to balance the workload of large queries and high-transaction volumes across multiple disks.

Parallel database query (PDQ) is an Informix database server feature that can improve performance dramatically when the database server processes queries initiated by decision-support applications. PDQ features allow the database server to distribute the work for one aspect of a query among several processors. For example, if a query requires an aggregation, the database server can distribute the work for the aggregation among several processors. PDQ also includes tools for memory-resource management.

PDQ delivers maximum performance benefits when the data that is being queried is in fragmented tables. For information on how to use PDQ and fragmentation for maximum performance, refer to your refer to your *Performance Guide*.

What Is Fragmentation?

Fragmentation is a database server feature that enables you to define groups of rows or index keys within a table according to some algorithm or *scheme*. You can store each group or *fragment* (also referred to as *partitions)* in a separate dbspace that is associated with a specific physical disk. You create the fragments and assign them to dbspaces with SQL statements.

From the perspective of an end user or client application, a fragmented table is identical to a nonfragmented table. Client applications do not require any modifications to allow them to access the data that is contained in fragmented tables.

The database server stores the location of each table and index fragment, along with other related information, in the system catalog table named **sysfragments**. You can use this table to access information about your fragmented tables and indexes. For the complete listing of the information in this system catalog table, refer to the *Informix Guide to SQL: Reference*.

Because the database server has information on which fragments contain which data, the database server can route client requests for data to the appropriate fragment without accessing irrelevant fragments, as Figure 15-1 illustrates. For more information on fragment elimination, refer to your *Performance Guide*.

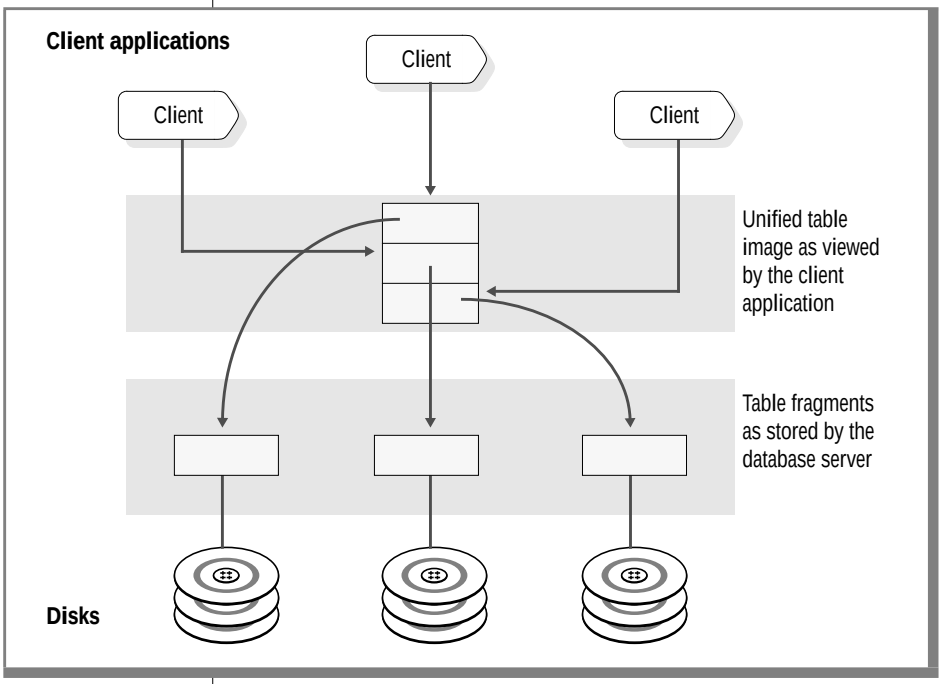


Figure 15-1
Routing Client Requests To The Appropriate Table Fragments

The following sections cover these fragmentation topics:

- Fragmentation goals
- Fragmentation strategies
- Summary of SQL statements for fragmentation

Fragmentation Goals

Consider fragmenting your tables if you have at least one of the following goals:

- Improved single-user response time

  To improve the performance of individual queries, use fragmentation with parallel database query (PDQ) to scan in parallel fragments that are spread across multiple disks.

- Improved concurrency

  Fragmentation can reduce contention for data that is located in large tables that are used by multiple queries and OLTP applications. Fragmentation reduces contention because each fragment resides on a separate I/O device, and the database server directs queries to the appropriate fragment.

- Improved availability

  If a fragment becomes unavailable, the database server can still access the remaining fragments.

- Improved backup-and-restore characteristics

  Fragmentation gives you a finer backup-and-restore granularity. This granularity can reduce the time that is required for backup-and-restore operations. In addition, you can improve the performance of backup-and-restore operations if you use ON-Bar or ON-Archive to perform these operations in parallel.

Each of the preceding goals has its own implications for the fragmentation strategy that you ultimately implement. "Fragmentation Strategies" on page 15-6 discusses these issues. Your primary fragmentation goal determines, or at least influences, how you implement your fragmentation strategy.

In deciding whether to use fragmentation to meet any of the preceding goals, keep in mind that fragmentation requires some additional administration and monitoring activity. For more information about fragmentation, refer to your *Performance Guide*.

Whose Responsibility Is Fragmentation?

Some overlap exists between the responsibilities of the database server administrator and those of the DBA (database administrator) with respect to fragmentation. The DBA creates the database schema. This schema can include table fragmentation. The database server administrator, on the other hand, lays out the disk space and creates the dbspaces where the fragmented tables reside. Because these responsibilities cannot be performed in isolation, implementing fragmentation requires a cooperative effort between the database server administrator and the DBA.

Fragmentation Strategies

A fragmentation strategy consists of two parts:

- A distribution scheme

  The scheme that you use to group rows or index keys into fragments is called the *distribution scheme*. You specify the distribution scheme in the FRAGMENT BY clause of the CREATE TABLE, CREATE INDEX, or ALTER FRAGMENT statement.

- The set of dbspaces in which you locate the fragments

  You specify the set of dbspaces in the IN clause of these SQL statements.

The database server supports the following distribution schemes:

- **Round-robin.** This type of fragmentation places rows one after another in fragments, rotating through the series of fragments to distribute the rows evenly.

  For INSERT statements, the database server uses a hash function on a random number to determine the fragment in which to place the row. For INSERT cursors, the database server places the first row in a random fragment, and the second and subsequent rows are assigned to fragments in sequence. If one of the fragments is full, that fragment is skipped.

- **Expression-based.** This type of fragmentation puts rows into fragments based on a *fragmentation expression* that you specify. This expression defines criteria, or rules, for assigning a set of rows to each fragment. The expression can take the form of a range or some other arbitrary rule. You can specify a *remainder fragment* that holds all rows that do not match the criteria for any other fragment, although a remainder fragment reduces the efficiency of the expression-based distribution scheme.

Fragmenting Tables

Formulating a fragmentation strategy for a table requires you to make the following decisions:

1. Decide what your primary fragmentation goal is.

   Your fragmentation goals depend, to a large extent, on the types of applications that access the table.

2. Decide how the table should be fragmented.

   You must make the following decisions:

   - Whether to fragment the table data, the table index, or both

     This decision is usually based on your primary fragmentation goal.

   - What is the ideal distribution of rows or index keys for the table

     This decision is also based on your primary fragmentation goal.

3. Decide on a distribution scheme.
4. To complete the fragmentation strategy, you must decide on the number and location of the fragments.

For more information on the decisions that you must make to formulate a fragmentation strategy, see the *Informix Guide to Database Design and Implementation*. For information on optimizing the performance of your fragmentation scheme, refer to your *Performance Guide*.

Fragmenting a Temporary Table

Just as you fragment permanent tables, you also can fragment an explicit temporary table across multiple disks.

To create a temporary, fragmented table, use the TEMP keyword of the CREATE TABLE statement. You can specify what distribution scheme and which dbspaces to use for the temporary table. For more information on the types of temporary tables, refer to "Temporary Tables" on page 13-31.

You can define your own fragmentation strategy for an explicit temporary table, or you can let the database server dynamically determine the fragmentation strategy. For more information, refer to your *Performance Guide*.

Fragmenting Table Indexes

You can fragment both table data and table indexes. When you create an index, you can:

- create an *attached index* by omitting the storage specification from the CREATE INDEX statement.

  When you do, the attached index takes on the same fragmentation strategy as the table. Each fragment of an attached index resides in the same dbspace as the corresponding table data.

  You create an attached index by omitting the FRAGMENT BY and IN clauses from the CREATE INDEX statement.

```
CREATE TABLE tb1 (a int)
    FRAGMENT BY EXPRESSION
        (a >= 0 and a < 5) IN dbspace1,
        (a >= 5 and a < 10) IN dbspace2
:
;
CREATE INDEX idx1 ON tb1(a);
```

- create a *detached index* by including an explicit storage specification in the CREATE INDEX statement.

  When you do, the detached index uses its own fragmentation strategy, which can differ from that of the table. A fragment in a detached index can reside in a different dbspace than the corresponding table data.

  You can use only the expression-based distribution scheme to fragment an index. You cannot use the round-robin distribution scheme for an index. For more information on the CREATE INDEX statement, refer to the *Informix Guide to SQL: Syntax*.

Fragmenting table data and table indexes can greatly affect performance. For detailed information on fragmenting table data and table indexes, see your *Performance Guide*.

Using SQL Statements to Perform Fragmentation Tasks

To perform most fragmentation tasks, you use appropriate SQL statements. Figure 15-2 lists the fragmentation tasks and the SQL statements to accomplish these tasks.

For details on how to accomplish these fragmentation tasks, refer to the *Informix Guide to Database Design and Implementation*. For the syntax of these SQL statements, refer to the *Informix Guide to SQL: Syntax*.

Figure 15-2
Fragmentation Tasks and Corresponding SQL Statements

Fragmentation Task	SQL Statements
Creating a new fragmented table	CREATE TABLE statement, FRAGMENT BY clause
Creating a fragmented table from a single nonfragmented table	ALTER FRAGMENT statement, INIT clause
Creating a fragmented table from more than one nonfragmented table	ALTER FRAGMENT statement, ATTACH clause
Modifying distribution scheme for a fragmented table	ALTER FRAGMENT statement, MODIFY clause

(1 of 2)

Fragmentation Task	SQL Statements
Adding a fragment to a table	ALTER FRAGMENT statement, ADD clause
Removing a fragment from a table	ALTER FRAGMENT statement, DROP clause
Reinitializing a fragmentation scheme	ALTER FRAGMENT statement, INIT clause
Converting a fragmented table to a non-fragmented table	ALTER FRAGMENT statement, INIT clause
Creating a fragmented index	CREATE INDEX statement
Adding an explicit rowid column to a fragmented table	ALTER TABLE statement, ADD ROWID clause

(2 of 2)

What Is PDQ?

PDQ refers to the techniques that the database server can use to distribute the execution of a single query over several processors. The database server can also use PDQ for queries that consume large quantities of non-CPU resources, in particular large quantities of memory and many disk scans.

A query that is processed with PDQ techniques is called a *PDQ query*. When the database server processes a PDQ query, it first divides the query into subplans. The database server then allocates the subplans to a number of threads that process the subplans in parallel. Because each subplan represents a smaller amount of processing time when compared to the original query, and because each subplan is processed simultaneously with all other subplans, the database server can drastically reduce the time that is required to process the query. Figure 15-3 illustrates this concept.

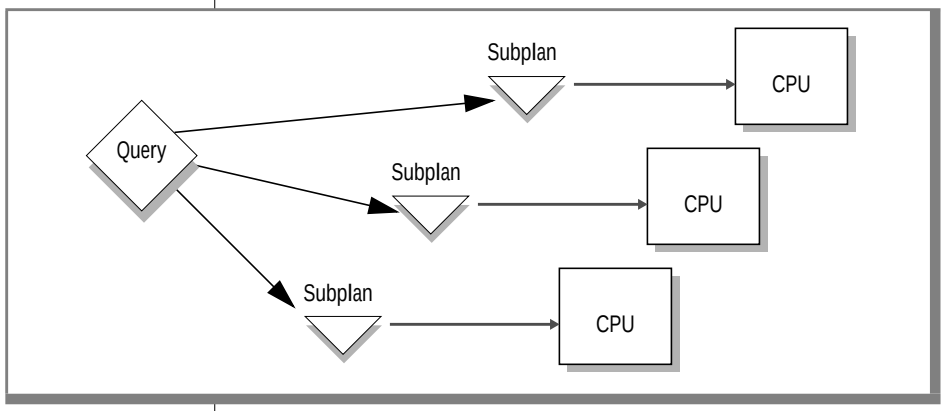


Figure 15-3
Parallel Database Query

PDQ consists of five principal components:

- Parallel scan
- Parallel join
- Parallel sort
- Parallel aggregate
- Parallel group

High Degree of Parallelism

The degree of parallelism for a query refers to the number of subplans that the database server executes in parallel to run the query. For example, a two-table join that six threads execute (with each thread executing one sixth of the required processing) has a higher degree of parallelism than one that two threads execute.

The database server determines the best degree of parallelism for each component of a PDQ query, based on values set by the database administrator, by the user, and by the client application, as well as various internal considerations such as the number of available virtual processors (VPs), the fragmentation of the tables that are being queried, the complexity of the query, and so on.

At times, the database server uses the components themselves in parallel. For example, consider what occurs when the database server must process a complex join. First, the database server scans the data in parallel. As soon as it has scanned sufficient data to begin the join, it does so. Just after the join begins, the database server begins the sort and performs other required processing until the full join is completed.

You gain the most benefit from PDQ when you use fragmented tables on a multiprocessor computer. However, PDQ can provide performance gains even with nonfragmented tables on a uniprocessor computer.

PDQ provides performance advantages on both uniprocessor computers and multiprocessor computers.

- On a uniprocessor computer, PDQ allows the database server to submit I/O requests to multiple disks in parallel and to take full advantage of the memory on the computer.
- On a multiprocessor computer, PDQ distributes the execution of a query across available processors.

If you want to derive the full performance benefits that PDQ can offer, Informix recommends that you run PDQ on a multiprocessor computer.

When Should You Use PDQ?

Applications that access data stored in a relational database can be divided into the following two types:

- On-line transaction-processing (OLTP) applications
- Decision-support applications

The complex queries that are typical of decision-support applications can benefit from PDQ.

The next sections describe the characteristics of OLTP and decision-support applications.

OLTP Applications

OLTP applications are characterized by quick, indexed access to a small number of data items. An order-entry system is an example of a typical OLTP system. The transactions handled by OLTP applications are usually simple and predefined.

OLTP applications can be characterized as follows:

- Simple transactions that involve small amounts of data
- Indexed access to data
- Many users
- Frequent requests
- Very fast response times

Processing OLTP Queries

The default behavior of the database server is ideal for OLTP transactions, optimizing performance for short transactions that require rapid response times. All queries have the same priority for CPU, memory, and disk I/O.

Queries that require quick response and generate only a small amount of information should *not* use PDQ. For example, the following queries should not use PDQ:

- Do we have a hotel room available in Berlin on December 8?
- Does the store in Mill Valley have green tennis shoes in size 4?

Warning: *Users must not set the* ***PDQPRIORITY*** *environment variable to a non-zero value for OLTP queries.*

The impact of PDQ on OLTP queries can be dramatic. One PDQ parameter limits the number of simultaneous queries the database server can perform. Suppose the number of simultaneous queries is set to 4. If another query requests service, it must wait until one of the previous four queries finishes. If the four queries are decision-support queries, the delay could be several minutes. Typical OLTP queries must be processed immediately.

Decision-Support Applications

Decision-support applications provide information for strategic planning, decision making, and report preparation. Decision-support applications frequently generate queries that require the database server to scan entire tables and manipulate large amounts of data. These queries can require operations such as multiple joins, temporary tables, and hundreds, if not thousands, of calculations. For example, the following queries should use the PDQ features of the database server:

- Based on the predicted number of housing starts, the known age of existing houses, and the observed roofing choices for houses in different areas and price ranges, what roofing materials should we order for each of our regional distribution centers?
- How does the cost of health-care plan X compare with the cost of health-care plan Y, considering the demographic profile of our company? Would plan X be better for some regions and plan Y for others?

Such operations require large amounts of data and large amounts of memory. As a result, the execution times for decision-support applications are far longer than the execution times required for typical OLTP applications. Other typical decision-support applications include payroll, inventory reporting, and end-of-period accounting reports. These applications are frequently executed in a batch environment.

Queries that contain one or more of the following operations require large quantities of memory:

- Merge joins
- Hash joins
- Sorting
- Groups

Other factors can also influence how the database server allocates resources to a query. Consider the following SELECT statement:

```
SELECT col1, col2 FROM table1 ORDER BY col1
```

If no indexes exist on **table1**, a sort is required, and hence the database server must allocate memory and temporary disk space to sort the query. However, if column **col1** is indexed, the query does not require these resources.

Decision-support applications have the following characteristics:

- Complex queries that involve large amounts of data
- Large memory requirements
- Few users
- Periodic requests
- Relatively long response times

When both OLTP and decision-support queries are running on the same computer, Dynamic Server must balance its resources so that all users receive the best possible performance.

The database server uses the *PDQ priority* value of a query to determine when to use PDQ to process a query in parallel. For more information on PDQ priority, refer to "PDQ Priority" on page 15-17.

Processing Decision-Support Queries

If a user sets the PDQ priority greater than 0 for a given query, the database server treats the query as a PDQ query and controls the PDQ resources allocated to it. PDQ resources include memory, CPU VPs, disk I/O, and scan threads. The database server determines the degree to which it will parallelize the query based on the following factors:

- The value of PDQ priority
- The availability of computer-system resources (CPUs, memory, and disk I/O)
- The value of parameters (NUMCPUVPS, DS_TOTAL_MEMORY, and so forth) set by the database server administrator

The parameters that control PDQ resources are discussed in the following section.

How Does the Database Server Allocate Resources with PDQ?

This section discusses the tools that you can use to balance resource use.

Parameters Used for Controlling PDQ

Figure 15-4 summarizes the configuration parameters, environment variables, and the SQL statement that control how the database server allocates resources to PDQ. The value set by the SQL statement supersedes values set by the environment variables, and values set by environment variables supersede values set by configuration parameters.

Figure 15-4
Parameters Used for Controlling PDQ

Configuration Parameters	Environment Variables	SQL Statements	Purpose of Parameter
DS_MAX_QUERIES			Maximum number of PDQ queries that can be active at any one time
DS_MAX_SCANS			Maximum number of PDQ scan threads that can execute concurrently
DS_TOTAL_MEMORY			Maximum amount of memory that can be allocated for use by PDQ
MAX_PDQPRIORITY			Percentage of user's requested PDQ priority value that the database server grants
OPTCOMPIND	**OPTCOMPIND**		Indicate a preferred join type to the query optimizer
	PDQPRIORITY	SET PDQPRIORITY	Request priority and percentage of PDQ resources for an application or a specific query

For more information on PDQ parameters, refer to the sources shown in the following table.

Parameter	Refer To
MAX_PDQPRIORITY	page 33-52 and your *Performance Guide*
DS_MAX_QUERIES	page 33-26 and your *Performance Guide*
DS_MAX_SCANS	page 33-27 and your *Performance Guide*
DS_TOTAL_MEMORY	page 33-28 and your *Performance Guide*
OPTCOMPIND	page 33-68 and your *Performance Guide*

PDQ Priority

Users specify what part of database-server resources to devote to PDQ queries by setting the **PDQPRIORITY** environment variable or using the SET PDQPRIORITY SQL statement. Users can use the SQL statement SET PDQPRIORITY to control the parallelism of individual SQL statements and the **PDQPRIORITY** environment variable to control the resources allocated to an individual user.

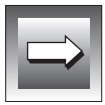
Users can set PDQ priority to OFF, LOW, or HIGH (values 0, 1, 100, respectively) or integer values between 0 and 100. The values 0 and 1 have special meanings:

- A PDQ priority value of 0 (the default) means that the features of PDQ are not used.
- A PDQ priority value of 1 means to do parallel scans only.
- All other values (2 through 100) represent the percent of the available PDQ resources that the query requests during its execution.

A query running with a PDQ priority of 100 uses all available memory and scan resources. If PDQ priority is less than 100, the query consumes proportionally fewer resources. You can limit the percentage of PDQ resources that the database server actually allocates to users by setting the MAX_PDQPRIORITY configuration parameter.

The environment variable and the SQL statement use the same range of values for setting PDQ priority. In addition, the DEFAULT option for the SET PDQPRIORITY statement allows an application to revert to the value for PDQ priority as set by the environment variable, if any. DEFAULT is the symbolic equivalent of the -1 value.

For more information about the environment variable and the SQL statement, refer to the *Informix Guide to SQL: Reference* and the *Informix Guide to SQL: Syntax*, respectively.



Important: *The database server does not decide whether a query should be a PDQ query. If the PDQ priority of a query is greater than zero, the processing of the query is controlled by the PDQ parameters, even if the query does not use any parallel processing.*

How Does the Database Server Use PDQ?

This section describes the types of SQL operations that the database server processes in parallel and the situations that limit the degree of parallelism that the database server can use. In the following discussions, a *query* is any SELECT statement.

SQL Operations That Take Advantage of PDQ

The database server takes the following two steps to process the DELETE, INSERT, and UPDATE statements:

1. Fetch the qualifying rows.
2. Apply the action of deleting, inserting, or updating.

Parallel Delete

The database server performs the first step of a DELETE statement in parallel, with one exception; the database server does not process the first part of a DELETE statement in parallel if the targeted table has a referential constraint that can cascade to a child table.

Parallel Inserts

The database server performs the following types of inserts in parallel:

- SELECT...INTO TEMP inserts using explicit temporary tables
- INSERT INTO...SELECT inserts using implicit temporary tables

For information on implicit and explicit temporary tables, refer to Chapter 13, "Where Is Data Stored?" and the *Informix Guide to SQL: Syntax*.

Explicit Inserts Using SELECT...INTO TEMP

The database server can insert rows in parallel into explicit temporary tables that you specify in SQL statements of the form SELECT....INTO TEMP. For example, the database server can perform the inserts in parallel into the temporary table, **temp_table**, as shown in the following example:

```
SELECT * FROM table1 INTO TEMP temp_table
```

The database server performs this type of parallel insert provided that you set PDQ priority > 0 and DBSPACETEMP is set to a list of two or more dbspaces.

The first item, PDQ priority > 0, is a requirement that you must meet for any query that you want the database server to perform in parallel.

The second item, that DBSPACETEMP is set to a list of two or more dbspaces, is required because of the way that the database server performs the insert. To perform the insert in parallel, the database server first creates a fragmented temporary table. So that the database server knows where to store the fragments of the temporary table, you must specify a list of two or more dbspaces in the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable. In addition, you must set DBSPACETEMP to indicate storage space for the fragments *before* you execute the SELECT...INTO statement.

The database server performs the parallel insert by writing in parallel to each of the fragments in a round-robin fashion. Performance improves as you increase the number of fragments.

Implicit Inserts with INSERT INTO...SELECT

The database server can also insert rows in parallel into implicit tables that it creates when it processes SQL statements of the form INSERT INTO...SELECT. For example, the database server processes the following INSERT statement in parallel:

```
INSERT INTO target_table SELECT * FROM source_table
```

The target table can be either a permanent table or a temporary table.

The database server processes this type of INSERT statement in parallel only when the target tables meet the following criteria:

- The value of PDQ priority is greater than 0.
- The target table is fragmented into two or more dbspaces.
- The target table has no enabled referential constraints or triggers.
- The target table is not a remote table.
- In a database with logging, the target table does not contain filtering constraints.
- The target table does not contain columns of TEXT or BYTE data type.

The database server does not process parallel inserts that reference a stored procedure. For example, the database server never processes the following statement in parallel:

```
INSERT INTO table1 EXECUTE PROCEDURE ins_proc
```

Parallel Index Builds

Index builds can take advantage of PDQ and can be parallelized. The database server performs both scans and sorts in parallel for index builds. The following operations initiate index builds:

- CREATE INDEX
- Add a unique, primary key
- Add a referential constraint
- Enable a referential constraint

When PDQ is in effect, the scans for index builds are controlled by the PDQ configuration parameters described in "Parameters Used for Controlling PDQ" on page 15-16.

If you have a computer with multiple CPUs, the database server uses two sort threads to sort the index keys. The database server uses two sort threads during index builds without the user setting the **PSORT_NPROCS** environment variable.

SQL Operations That Do Not Use PDQ

The database server does not process the following types of queries in parallel:

- Queries started with an isolation mode of Cursor Stability

  Subsequent changes to the isolation mode do not affect the parallelism of queries already prepared. This situation results from the inherent nature of parallel scans, which scan several rows simultaneously.
- Queries that use a cursor declared as FOR UPDATE or with the WITH HOLD qualifier
- An UPDATE statement that has an *update* trigger that updates in the For Each Row section of the trigger definition
- Data definition language (DDL) statements

  For a complete list, see the *Informix Guide to SQL: Syntax*.

Update Statistics

The SQL UPDATE STATISTICS statement, which is not processed in parallel, is affected by PDQ because it must allocate the memory used for sorting. Thus the behavior of the UPDATE STATISTICS statement is affected by the memory management associated with PDQ.

Even though the UPDATE STATISTICS statement is not processed in parallel, the database server must allocate the memory that this statement uses for sorting.

Stored Procedures and Triggers

Statements that involve stored procedures are not executed in parallel. However, statements within procedures are executed in parallel.

When the database server executes a stored procedure, it does not use PDQ to process nonrelated SQL statements contained in the procedure. However, each SQL statement taken as an independent statement can be executed in parallel using intraquery parallelism when possible. As a consequence, you should limit the use of procedure calls from within data manipulation language (DML) statements if you want to exploit the parallel-processing abilities of the database server. For a complete list of DML statements, see the *Informix Guide to SQL: Syntax*.

The database server uses intraquery parallelism to process the statements in the body of an SQL trigger in the same way that it processes statements in stored procedures.

Correlated and Uncorrelated Subqueries

The database server does not use PDQ to process correlated subqueries. Only one thread at a time can execute a correlated subquery. While one thread executes a correlated subquery, other threads that request to execute the subquery are blocked until the first one completes.

For uncorrelated subqueries, only the first thread that makes the request actually executes the subquery. Other threads simply use the results of the subquery and can do so in parallel.

As a consequence, Informix strongly recommends that, whenever possible, you build your queries using joins rather than subqueries so that your queries can take advantage of PDQ.

Outer Index Joins

The database server reduces the PDQ priority of queries that contain OUTER index joins to LOW (if it is set to a higher value) for the duration of the query. If a subquery or a view contains OUTER index joins, the database server lowers the PDQ priority of only that subquery or view, not of the parent query or any other subquery.

Remote Tables

A remote table has all its fragments in the same remote database. Although the database server can process the data stored in a remote table in parallel, the data is communicated serially because the database server allocates a single thread to submit and receive the data from the remote table.

The database server lowers the PDQ priority of queries that require access to a remote database to LOW. In that case, all local scans are parallel, but all local joins and remote access are nonparallel.

Logging and Log Administration

Section V

Chapter

16

What Is Logging?

Which Database Server Processes Require Logging? 16-3

What Database Server Activity Is Logged?. 16-5
Activity That Is Always Logged 16-6
Activity Logged for Databases with Transaction Logging 16-7
Are Blobs Logged? 16-7

What Is Transaction Logging? 16-8
The Database-Logging Status 16-8
Unbuffered Transaction Logging 16-9
Buffered Transaction Logging 16-9
ANSI-Compliant Transaction Logging 16-10
Databases with Different Log-Buffering Status. 16-10
When to Use Transaction Logging 16-10
When to Buffer Transaction Logging 16-11
Who Can Set or Change Logging Status? 16-11

This chapter describes Informix database server logging. First, the chapter describes logging with respect to the database server functionality. It addresses the following questions:

- Which database server features require logging?
- What database server activity is logged?

Next, the chapter describes logging with respect to databases. You specify whether a database uses *transaction logging* and, if it does, what log-buffering mechanism it uses. The chapter addresses the following questions:

- What is the database logging status?
- When should transaction logging be used?
- When should buffered transaction logging be used?
- Who can set or change the database logging status?

Which Database Server Processes Require Logging?

As the database server operates—as it processes transactions, keeps track of data storage, ensures data consistency, and so on—it automatically generates *logical-log records* for some of the actions it takes. Most of the time, the database server makes no further use of the 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 log records are critical. The log records are at the heart of the database server data-recovery mechanisms.

Tip: *If you use ON-Bar for your backup and restore tool, see your "Backup and Restore Guide" or information on how to restore your data from a dbspace backup. If you use ON-Archive or* ***ontape*** *for your backup and restore tool, see your "Archive and Backup Guide."*

The database server stores the 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 off-line (*backed up*). The database server administrator keeps the off-line log records (in 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. For information about logical-log administration topics, see Chapter 18, "What Is the Logical Log?"

The database server uses logical-log records when it performs various functions that recover data and ensure data consistency, as follows:

- Fast recovery

  If the database server shuts down in an uncontrolled manner, the database server uses the log records to recover all transactions that occurred since the most-recent checkpoint—when all the data in shared memory and all the data on disk were the same (also known as *physically consistent*)—and to roll back any uncommitted transactions. The database server uses the log records in the second phase of 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 "Details of Fast Recovery" on page 22-5.)

- Transactions roll back

  If a database has transaction logging turned on (see "What Is Transaction Logging?" on page 16-8), and a transaction must be rolled back, the database server uses the log records to reverse the changes made on behalf of the transaction.

- Data restoration

  During a data restore, you combine the backup tapes of the logical-log files with the most-recent database server dbspace backup tapes to re-create the database server system up to the point of the most-recently backed-up logical-log record. After the dbspace backup tapes have been restored, the database server essentially uses the log records to reimplement all the logged activity since the last dbspace 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 from the transaction to roll back the transaction.

- Cascading deletes

  Cascading deletes on referential constraints use 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.

- 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 30-30.

- High-availability data replication (HDR)

  High-availability data replication uses logical-log records to maintain consistent data on two different database servers so that one of the database servers can be used quickly as a backup database server if the other fails. For a more detailed discussion of how high-availability data replication uses logical-log records, see "How Does High-Availability Data Replication Work?" on page 25-8.

What Database Server Activity Is Logged?

The database server does not generate log records for every operation because it does not need a record of every action.The database server needs log records only to perform the functions listed in "Which Database Server Processes Require Logging?" on page 16-3. Also, the space required to store a record of everything the database server did would quickly be overwhelming.

The logical-log records themselves are variable length. This arrangement increases the number of 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.

Two 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

The following sections explain the two different types of activity. For more information on the format of logical-log records, see Chapter 37, "Interpreting Logical-Log Records."

Activity That Is Always Logged

Some database operations always generate logical-log records, even if none of the databases on the database server use transaction logging. See "What Is Transaction Logging?" on page 16-8. These operations are as follows:

- SQL data definition statements for all databases:

ALTER INDEX	CREATE VIEW
ALTER TABLE	DROP INDEX
CREATE DATABASE	DROP PROCEDURE
CREATE INDEX	DROP SYNONYM
CREATE PROCEDURE	DROP TABLE
CREATE SCHEMA	DROP TRIGGER
CREATE SYNONYM	DROP VIEW
CREATE TABLE	RENAME COLUMN
CREATE TRIGGER	RENAME TABLE

- Dbspace backup events
- Checkpoint events
- Administrative changes to the database server configuration

  This category includes changes to the number and location of chunks, dbspaces, and blobspaces.
- Allocation of new extents to tables
- A change to the logging status of a database

Activity Logged for Databases with Transaction Logging

If a database uses transaction logging, all SQL data manipulation statements (DML), except SELECT, against that database generate one or more log records. These statements are as follows:

- DELETE
- INSERT
- LOAD
- SELECT INTO TEMP
- UNLOAD
- UPDATE

If these statements are rolled back, the rollback also generates log records.

Are Blobs Logged?

Blob data (TEXT and BYTE data types) is potentially too voluminous to include in a logical-log record. If blob data were always included, the many kilobytes of data per blob could overwhelm the space allocated for the logical log. However, not all blobs are that large, and not every blob would overwhelm the logical log.

The database server assumes that you designed your databases so that smaller blobs are stored in dbspaces, and larger blobs are stored in blobspaces. For information on how to locate blob data, see your *Performance Guide*. Based on this assumption, the database server takes the following action:

- The database server includes blob data in log records for blobs stored in dbspaces.
- The database server does not include blob data in log records for blobs stored in blobspaces.

The database server still needs access to the blobs in blobspaces in order to fulfill the goals of logging, explained in "Which Database Server Processes Require Logging?" on page 16-3. Consider how a blobspace operation can be rolled back or used in fast recovery if the logical log does not contain a copy of the data that was originally inserted. The answer is that the log keeps a pointer to the location of the actual data. "Blobspace Logging" on page 18-27 describes this mechanism.

What Is Transaction Logging?

A database is said to *have* or *use* transaction logging, or have transaction logging *turned on*, when SQL data manipulation statements in a database generate logical-log records.

The database-logging *status* indicates whether a database uses transaction logging. You can set the database-logging status (turn on transaction logging, for example) when you create the database. You can also change the database status (turn off transaction logging, for example) after the database is created. For information on how to change the database-logging status, see "Who Can Set or Change Logging Status?" on page 16-11 and Chapter 17, "Managing Database-Logging Status."

Even if you turn off transaction logging for all databases in a database server, the database server always logs some events, as listed in "Activity That Is Always Logged" on page 16-6.

The Database-Logging Status

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 Databases" on page 29-43. The database-logging status indicates any of the following types of logging:

- No logging
- Unbuffered transaction logging

- Buffered transaction logging
- ANSI-compliant transaction logging

The last three items in this list refer to different log-buffering mechanisms. As explained in "How the Database Server Uses Shared Memory" on page 11-6, information that the database server manages passes through shared memory to disk. Logical-log records are no exception. Before the database server writes logical-log records to the logical log, which is on disk, the records must pass through shared memory. They do this through the logical-log buffers, explained in "Flushing the Logical-Log Buffer" on page 11-53.

In one sense, all the database server logging is buffered because all 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.

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 before the COMMIT statement (and before the PREPARE statement for distributed transactions) returns to the application. 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 used buffered logging.

Buffered Transaction Logging

If transactions are 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.

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 all use unbuffered transaction logging, enforcing the ANSI rules for transaction processing.

Databases with Different Log-Buffering Status

All databases 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 database(s) with unbuffered logging complete.

When to Use Transaction Logging

You must use transaction logging with a database to take advantage of any of the features listed in "Which Database Server Processes Require Logging?" on page 16-3.

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of the 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 log records of their transactions until you reinitiate logging, which must wait for a level-0 dbspace backup.

If you use a distributed environment, the logging status of the databases must be the same (all buffered, all unbuffered, all ANSI compliant, or all without transaction logging). If one of the databases in a distributed query uses transaction logging, the others must also.

When to Buffer Transaction Logging

If a database does not use logging, you do not need to consider whether buffered or unbuffered logging is more appropriate.

ANSI-compliant databases always use unbuffered logging. You cannot change the buffering status of ANSI-compliant databases.

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.

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.

Who Can Set or Change Logging Status?

The user who creates a database with the CREATE DATABASE statement establishes the logging status for that database. If the CREATE DATABASE statement does not specify a logging status, the database is created without logging. For more information on the CREATE DATABASE statement, see the *Informix Guide to SQL: Syntax*.

Only the database server administrator can change the logging status, which is described in Chapter 17, "Managing Database-Logging Status." End users can switch from unbuffered to buffered (but not ANSI-compliant) transaction logging, and from buffered to unbuffered transaction 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 *Informix Guide to SQL: Syntax*.

Chapter

17

Managing Database-Logging Status

About Changing Logging Status 17-3

Modifying Database-Logging Status with ON-Archive 17-5
- Turning On Transaction Logging with ON-Archive 17-5
- Canceling a Logging Operation with ON-Archive 17-6
- Ending Logging with ON-Archive 17-6
- Changing Buffering Status with ON-Archive 17-7
- Making a Database ANSI Compliant with ON-Archive 17-7

Modifying Database-Logging Status with ontape 17-7
- Turning On Transaction Logging with ontape 17-7
- Ending Logging with ontape 17-8
- Changing Buffering Status with ontape 17-8
- Making a Database ANSI Compliant with ontape 17-9

Modifying Database Logging Status with ON-Monitor 17-9

This chapter provides instructions on changing the database-logging status for databases that the database server manages. As a database server administrator, you can alter the logging status of a database as follows:

- Add transaction logging (buffered or unbuffered) to a database.
- End transaction logging for a database.
- Change transaction logging from buffered to unbuffered.
- Change transaction logging from unbuffered to buffered.
- Make a database ANSI compliant.

For information about database-logging status, and discussions of when to use transaction logging and when to buffer transaction logging, see Chapter 16, "What Is Logging?" To find out the current logging status of a database, see "Monitoring Databases" on page 29-43.

About Changing Logging Status

Figure 17-1 on page 17-4 shows the database-logging-status transitions that the database server administrator can perform and indicates whether they take place immediately or require a level-0 dbspace backup.

When you add logging (in any form) to a database that formerly did not use transaction logging, the change is not complete until you perform a level-0 dbspace backup on all the dbspaces and blobspaces that contain data in the database.

To change the logging status of ANSI-compliant databases, unload and reload the data.

Figure 17-1
Logging-Status Transitions

Converting from:	Converting to:			
	No logging	**Unbuffered logging**	**Buffered logging**	**ANSI compliant**
No logging	Not applicable	Level-0 dbspace backup (of affected dbspaces)	Level-0 dbspace backup (of affected dbspaces)	Level-0 dbspace backup (of affected dbspaces)
Unbuffered logging	Immediate	Not applicable	Immediate	Immediate
Buffered logging	Immediate	Immediate	Not applicable	Immediate
ANSI compliant	Illegal	Illegal	Illegal	Not applicable

Some general points about changing the database-logging status follow:

- Use the same backup tool (ON-Archive, ON-Bar, or **ontape**) to add logging and create the dbspace backup.

  If you use ON-Bar for your backup and restore tool, see your *Backup and Restore Guide* for information on how to restore your data from a dbspace backup. If you use ON-Archive or **ontape** for your backup and restore tool, see your *Archive and Backup Guide*.

- To make any change in the logging status of a database, no users can access the database. Once you start to make the change, the database server places an exclusive lock on the database to prevent other users from accessing the database.

- A database remains locked to users until the logging-mode change is complete. Some changes occur immediately, but if you add logging to a database that formerly did not have logging, the change is not complete until the next level-0 dbspace backup of all the dbspaces that contain data for the database.

- When you use ON-Archive, the database server must be in on-line mode in order to make changes. For **ontape** or ON-Monitor, the database server can be in either on-line or quiescent mode. (ON-Monitor is available only on UNIX.)
- If a failure occurs during a logging-mode change, check the flags for the database after you restore the database server (or dbspace). For more information, see "Monitoring Databases" on page 29-43.
- Once you choose either buffered or unbuffered logging, you can change from one logging status to the other in an application that uses the SQL statement SET LOG. This change lasts for the duration of the session.

Modifying Database-Logging Status with ON-Archive

If you use ON-Archive as your backup tool, use the MODIFY / DBLOGGING command to modify database-logging status. Reference information for ON-Archive is in your *Archive and Backup Guide*.

Turning On Transaction Logging with ON-Archive

Before you make this change, read "About Changing Logging Status" on page 17-3.

To add buffered logging to a database called **stores7** with ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores7/MODE=BUFFERED
```

To add unbuffered logging, use UNBUFFERED as the parameter with the MODE qualifier.

Canceling a Logging Operation with ON-Archive

After you turn on logging for a database with ON-Archive, you can turn logging off again (and unlock the database) before the next level-0 dbspace backup of all the dbspaces in the database.

To determine if a database is locked because logging has been turned on but the database has not yet been backed up, issue the following query to the **sysmaster** database:

```
SELECT name FROM sysdatabases WHERE flags > 255
```

To turn logging off for a database called **stores7** with ON-Archive (after it has been turned on, but before the change is completed by a level-0 dbspace backup), use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores7/MODE=CANCELCHANGE
```

The change takes place immediately; the database is unlocked. You can cancel any number of commands to add transaction logging to a database with the same command.

Ending Logging with ON-Archive

Before you make this change, read "About Changing Logging Status" on page 17-3.

To end logging for a database called **stores7** with ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores7/MODE=NOLOGGING
```

Changing Buffering Status with ON-Archive

Before you make this change, read "About Changing Logging Status" on page 17-3.

To change the buffering status for a database called **stores7** with transaction logging using ON-Archive, use one of the following commands, depending on whether or not you want the database to have buffered logging:

```
Onarchive> MODIFY/DBLOGGING=stores7/MODE=BUFFERED
Onarchive> MODIFY/DBLOGGING=stores7/MODE=UNBUFFERED
```

Making a Database ANSI Compliant with ON-Archive

Before you make this change, read "About Changing Logging Status" on page 17-3. Once you change the logging status to ANSI compliant, you cannot easily change it again. You must unload and reload the data.

To make a database called **stores7** ANSI compliant with ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores7/MODE=ANSI
```

Modifying Database-Logging Status with ontape

If you use **ontape** as your backup tool, you can use **ontape** to change the logging status of a database. Reference information for **ontape** is in your *Archive and Backup Guide*.

Turning On Transaction Logging with ontape

Before you make this change, read "About Changing Logging Status" on page 17-3.

You add logging to a database with **ontape** at the same time that you create a level-0 dbspace backup.

For example, to add buffered logging to a database called **stores7** with **ontape**, execute the following command:

```
% ontape -s -B stores7
```

To add unbuffered logging to a database called **stores7** with **ontape**, execute the following command:

```
% ontape -s -U stores7
```

In addition to turning on transaction logging, these commands create full-system dbspace backups. When **ontape** prompts you for a backup level, specify a level-0 dbspace backup.

Tip: *With **ontape**, you must perform a level-0 backup of all dbspaces. The ON-Archive utility permits greater archiving granularity, so you can restrict the dbspace backup to only those dbspaces that contain the database data.*

Ending Logging with ontape

Before you make this change, read "About Changing Logging Status" on page 17-3.

To end logging for a database called **stores7** with **ontape**, execute the following command:

```
% ontape -N stores7
```

Changing Buffering Status with ontape

Before you make this change, read "About Changing Logging Status" on page 17-3.

To change the buffering status from buffered to unbuffered logging on a database called **stores7** using **ontape** without creating a dbspace backup, execute the following command:

```
% ontape -U stores7
```

To change the buffering status from unbuffered to buffered logging on a database called **stores7** using **ontape** without creating a dbspace backup, execute the following command:

```
% ontape -B stores7
```

Making a Database ANSI Compliant with ontape

Before you make this change, read "About Changing Logging Status" on page 17-3. Once you change the logging status to ANSI compliant, you cannot easily change it again. You must unload and reload the data. For information about how to load and unload data, see the *Informix Migration Guide*.

To make databases ANSI compliant, you use different commands for databases that already use transaction logging and for those that do not use transaction logging.

To make a database called **stores7**, which already uses transaction logging (either unbuffered or buffered), into an ANSI-compliant database with **ontape**, execute the following command:

```
% ontape -A stores7
```

To make a database called **stores7**, which does not already use transaction logging, into an ANSI-compliant database with **ontape**, execute the following command:

```
% ontape -s -A stores7
```

In addition to making a database ANSI compliant, this command also creates a dbspace backup at the same time. Specify a level-0 dbspace backup when you are prompted for a level.

UNIX

Modifying Database Logging Status with ON-Monitor

Before you make any changes, read "About Changing Logging Status" on page 17-3.

You can use ON-Monitor to make any logging-status changes that can occur immediately. If you want to add logging to (or make ANSI compliant) a database that does not use logging, you cannot use ON-Monitor; you must use ON-Archive or **ontape**.

To change the logging status for a database from within ON-Monitor, select the Logical-Logs menu, Databases option.

Use the Arrow keys to select the database from which you want to remove logging. Press CTRL-B or F3.

When the logging-options screen appears, ON-Monitor displays the current log status of the database. Use the Arrow keys to select the status you want. Press CTRL-B or F3.

Chapter

18

What Is the Logical Log?

What Is the Logical Log? 18-3

What Is a Logical-Log File? 18-4

How Big Should the Logical Log Be?. 18-5
- Performance Considerations 18-5
- Long-Transaction Consideration 18-6
- Logical-Log Size Guidelines 18-6
- Determining the Size of the Logical Log 18-7

Preserving Log Space for ON-Archive Tasks 18-8
- Enabling the Logs-Full High-Water Mark 18-9
- Emergency Log Backup 18-9
  - Building the System-Monitoring Interface 18-10
  - Recovery. 18-10
  - Small Logs, Many Users 18-10
  - Administrative Activity When Logs Need Backing Up 18-10

What Should Be the Size and Number of Logical-Log Files?. 18-11

Where Should Logical-Log Files Be Located? 18-12

How Are Logical-Log Files Identified? 18-12

What Are the Status Flags of Logical-Log Files? 18-13

Point-In-Time Recovery 18-15

Why Do Logical-Log Files Need to Be Backed Up? 18-15

When Are Logical-Log Files Freed? 18-16
When Does the Database Server Attempt to Free a Log File? . . . 18-16
What Happens If the Next Logical-Log File Is Not Free? 18-16
Avoiding Long Transactions 18-18
Factors That Influence the Rate at Which Logical-Log Files Fill 18-18
Factors That Prevent Closure of Transactions 18-19
Setting High-Water Marks 18-20

What Are the Logical-Log Administration Tasks Required for Blobspaces? 18-21
Switching Logical-Log Files to Activate Blobspaces 18-21
Switching Logical-Log Files to Activate New Blobspace Chunks 18-22
Backing Up Logical-Log Files to Free Blobpages 18-22
Why Do You Have to Back Up Logical-Log Files to Free Blobpages? 18-22

What Is the Logging Process? 18-25
Dbspace Logging 18-25
Read Page into Shared-Memory Buffer Pool 18-25
Copy the Page Buffer into the Physical-Log Buffer 18-26
Read Data into Buffer and Create Logical-Log Record 18-26
Flush Physical-Log Buffer to the Physical Log 18-26
Flush Page Buffer 18-27
Flush Logical-Log Buffer 18-27
Blobspace Logging 18-27

As the database server administrator, you have responsibilities to configure and manage the logical log. These responsibilities include the following tasks:

- Allocating an appropriate amount of disk space for the logical log
- Choosing an appropriate number of logical-log files
- Monitoring the logical-log file status
- Backing up the logical-log files to tape

The information in this chapter will help you perform these tasks. In addition, this chapter gives background information on the database server logging process.

If you use ON-Bar as your backup and restore tool, see your *Backup and Restore Guide* for additional information about logical logs. If you use ON-Archive or **ontape** as your backup and restore tool, see your *Archive and Backup Guide*.

For information on how to perform other logical-log tasks, see Chapter 19, "Managing Logical-Log Files."

What Is the Logical Log?

To keep a history of database and database server changes since the time of the last dbspace backup, the database server generates and stores log records. The database server stores the log records in the logical log, which is made up of logical-log files. The log is called *logical* because the log records represent units of work related to the logical operations of the database server, as opposed to physical operations. At any time, the combination of database server backup tapes plus database server logical-log files contains a complete copy of your database server data.

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 or whether their transaction logging is buffered. For information on transaction logging, see Chapter 16, "What Is Logging?"

Most end users should not be concerned with the logical log and logical-log files. They might be concerned with the buffering status of a database during their transactions, or even occasionally with the transaction-logging status of a database, as explained in"Who Can Set or Change Logging Status?" on page 16-11.

Most of the administration of the logical log concerns the management of individual logical-log files, but one administrative task relates to the logical log as a whole: determining how much disk space to allocate to the logical log.

What Is a Logical-Log File?

Logical-log files are not files in the operating-system sense of the word *file*. Logical-log files are part of the disk space managed by the database server; each logical-log file is a separate allocation of disk space. Together, the logical-log files make up the logical log. You must always have at least three logical-log files.

The database server administrator needs to be concerned with the logical-log files that make up the logical log because, if the files are not managed properly, the database server can suspend processing and, in the worst case, shut down.

The database server administrator must choose an appropriate number, size, and physical location for logical-log files. The following sections discuss these topics:

- "What Should Be the Size and Number of Logical-Log Files?" on page 18-11
- "Where Should Logical-Log Files Be Located?" on page 18-12

The database server administrator must also ensure that the next logical-log file is always backed up and free. The following sections discuss this topic:

- "Why Do Logical-Log Files Need to Be Backed Up?" on page 18-15
- "When Are Logical-Log Files Freed?" on page 18-16

Some logical-log file administration tasks relate to managing blobspaces effectively, as discussed in "What Are the Logical-Log Administration Tasks Required for Blobspaces?" on page 18-21.

How Big Should the Logical Log Be?

In determining how much disk space to allocate, you must balance disk space and performance considerations. 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.

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 logical-log file backups and checkpoints, as follows:

- Logical-log file backups

  When the logical-log files that make up the logical log fill, you have to back them up. See "Why Do Logical-Log Files Need to Be Backed Up?" on page 18-15. The backup process can hinder transaction processing that involves data located on the same disk as the logical-log files. If enough logical-log disk space is available, however, you can wait for periods of low user activity before you back up the logical-log files.

- Checkpoints

  At least one checkpoint record must always be written to the logical log. If you need to free the logical-log file that contains the last checkpoint, the database server must write a new checkpoint record to the current logical-log file. See "When Are Logical-Log Files Freed?" on page 18-16. So if the frequency with which logical-log files are backed up and freed increases, the frequency at which checkpoints occur increases. Because checkpoints block user processing, this will have an adverse affect on performance. Because other factors (such as the physical-log size) also determine the checkpoint frequency, this effect might not be significant.

These performance considerations are related to how fast the logical log fills. The rate at which the logical log fills, in turn, depends on other factors such as the level of user activity on your system and the logging status of the databases. You need to tune the logical-log size, therefore, to find the optimum value for your system.

Long-Transaction Consideration

In addition to the performance considerations discussed in the previous section, you risk a long-transaction situation if logical-log disk space is insufficient. For more information on the long-transaction situation, refer to "Avoiding Long Transactions" on page 18-18.

Logical-Log Size Guidelines

Use the LOGSIZE configuration parameter to set the size of the logical log. It is difficult to predict how much logical-log space your database server system requires until it is fully in use. The following expression gives the *minimum* total-log-space configuration, in kilobytes, that Informix recommends:

```
LOGSIZE = (users * maxrows) * 512
```

Set *users* to the maximum number of users that you expect to access the database server concurrently. If you set the NETTYPE parameter, you can use the value that you assigned to the NETTYPE **users** field. If you configured more than one connection by setting multiple NETTYPE configuration parameters in your configuration file, sum the **users** fields for each NETTYPE, and substitute this total for *users* in the preceding formula.

For more information about LOGSIZE and NETTYPE, see Chapter 33, "Configuration Parameters."

You can increase the amount of space devoted to the logical log as necessary and in several ways. The easiest way is to add another logical-log file, as explained in "Adding a Logical-Log File" on page 19-4. To obtain better overall performance for applications that perform frequent updates of blobs in blobspaces, reduce the size of the logical log. For more information, see your *Performance Guide*.

Determining the Size of the Logical Log

The LOGFILES parameter is the number of logical-log files. 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 you add logical-log files that are not the size specified by LOGSIZE, you cannot use the (LOGFILES * LOGSIZE) expression to calculate the size of the logical log. Instead, you need to add the sizes of the individual logs file on disk. For information on how to access the size of logical-log files, see "Monitoring Logical-Log Files" on page 29-44. For more information about LOGFILES, see Chapter 33, "Configuration Parameters."

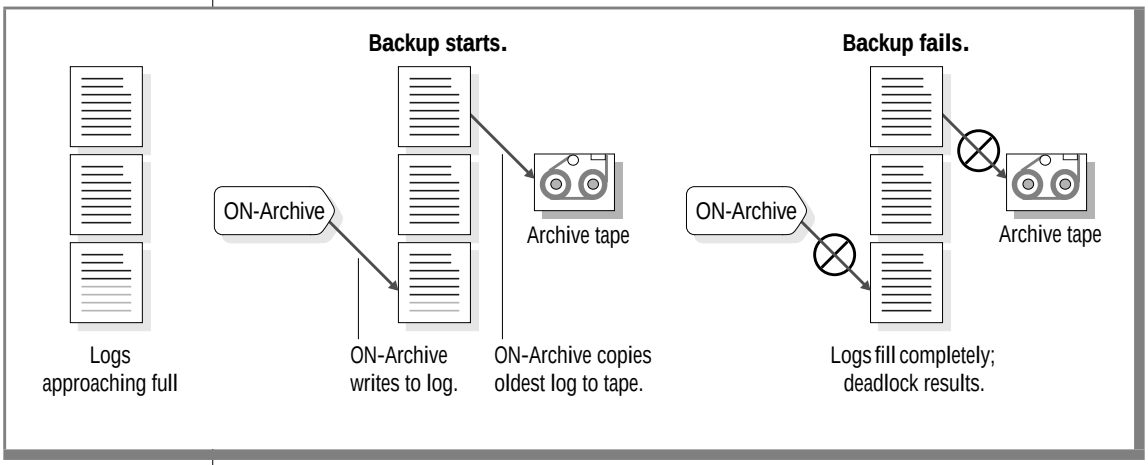
UNIX

Preserving Log Space for ON-Archive Tasks

If you use ON-Archive for your backup and restore tool, you can occasionally encounter a deadlock during peak activity on high-volume, on-line transaction-processing (OLTP) systems. The deadlock occurs when OLTP activity fills the logical log faster than ON-Archive can back up the logs to tape and free them.

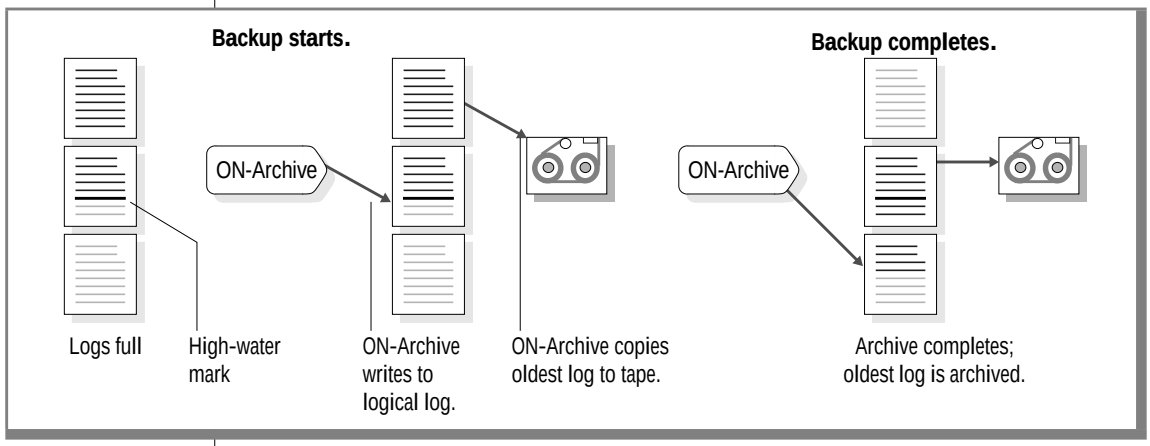
Prior to marking a backed-up log as free, the database server updates the ON-Archive catalog tables to record the occurrence of the logical-log backup. This update itself generates logical-log activity, leading to a possible deadlock as the logs continue to fill. When a deadlock of this type occurs, the database server administrator must use the emergency log backup procedure even though a continuous logical-log backup is already in progress. Figure 18-1 illustrates how a deadlock of this type occurs.

Figure 18-1
ON-Archive with High-Water Mark Off



The ON-Archive high-water-mark feature provides a solution for logical-log deadlocks of this type. When you enable this feature, the database server blocks OLTP activity when the next-to-last log fills, rather than the last log, as it usually does. In doing so, the database server preserves the last logical-log file for record logging generated by administrative activities such as a backup of the logical log. Figure 18-2 illustrates how the high-water mark feature prevents logical-log deadlocks.

Figure 18-2
ON-Archive with High-Water Mark On



For more information on the emergency backup procedure, see your *Archive and Backup Guide*.

Enabling the Logs-Full High-Water Mark

To enable the logs-full high-water mark, set the LBU_PRESERVE configuration parameter to 1. When you set LBU_PRESERVE to 1, the database server blocks DB-Access, ESQL/C, and all other clients from generating log records in the last logical-log file when the logs-full condition is reached. The default value of LBU_PRESERVE is 0, or off.

Whenever you change the value of LBU_PRESERVE, you must reinitialize shared memory for the change to take effect.

Emergency Log Backup

Although the logs-full high-water mark eliminates the need for emergency backup in the deadlock scenario described in the preceding sections, four known scenarios still require the database server administrators to use emergency log backup. Each case is examined in detail in the following sections.

Building the System-Monitoring Interface

A privileged client is responsible for building the system-monitoring interface (SMI). This client can potentially invade the last log file. If you do not configure sufficient log space or a sufficient number of log files, the privileged client might not succeed in building SMI without a log backup. This situation can cause the logical log to fill.

Recovery

When you start the database server after an uncontrolled shutdown, it needs log space to roll back any transactions that were uncommitted when the shutdown occurred. The threads that perform the recovery have privileges that allow them to use the last log file. Because of this privilege, the logical log might become full, but only in the unlikely case that the number and size of transactions open when the shutdown occurred exceed the size of the logical log.

Small Logs, Many Users

Perhaps you configure your logical log files as follows:

```
Logical Log Size < 2 * page_size * number of users
```

If all users enter transactions of maximum complexity, applications might invade the last log with OLTP activity. Only when you set the size of the log much smaller than two pages per user can a logs-full condition occur.

Administrative Activity When Logs Need Backing Up

Certain administrative clients have the privilege to invade the last logical-log file. The following list gives examples of such administrative clients:

- onspaces
- onparams
- oncheck
- ontape
- onmonitor
- ON-Archive
- oncatlgr
- onautovop
- ondatartr

Because these clients can invade the last logical log, certain circumstances might require you to perform an emergency log backup. For example, when the logical log approaches full, and you proceed to do large quantities of administrative work, you might need to perform an emergency log backup.

What Should Be the Size and Number of Logical-Log Files?

After you know how much disk space to allocate for the entire logical log, you can make decisions about how many log files you want, and of what size.

When you think about the *size* of the logical-log files, consider these points:

- The minimum size for a logical-log file is 200 kilobytes.
- The maximum size for a logical-log file is essentially unbounded.
- If your tape device is slow, ensure that logical-log files are small enough to be backed up in a timely fashion.
- Smaller log files mean smaller granularity of recovery because you potentially lose the last unbacked-up logical-log file if the disk that contains the logical-log files goes down.

When you think about the *number* of logical-log files, consider these points:

- You must always have at least three logical-log files.
- You should create enough logical-log files so that you can switch log files if needed without running out of free logical-log files.
- The number of logical-log files affects the frequency of logical-log backups and, consequently, the rate at which blobspace blobpages can be reclaimed. See "Backing Up Logical-Log Files to Free Blobpages" on page 18-22.
- The number of logical-log files cannot exceed the value of the ONCONFIG parameter LOGSMAX.

Where Should Logical-Log Files Be Located?

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 19-8.

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.

How Are Logical-Log Files Identified?

Each logical-log file, whether backed up to tape 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.

The actual disk space allocated for each logical-log file has an identification number known as the *logid*. For example, if you configure six logical-log files, these files have logid numbers one through six. As logical-log files are backed up and freed (see "Why Do Logical-Log Files Need to Be Backed Up?" on page 18-15), the database server reuses the disk space for the logical-log files. However, the database server continues to increment the unique ID numbers by one. Figure 18-3 on page 18-13 illustrates the relationship between the logid numbers and the unique ID numbers.

Figure 18-3
Logical-Log File-Numbering Sequence

Logid number	1st rotation unique ID number	2nd rotation unique ID number	3rd rotation unique ID number	4th rotation unique ID number
1	1	7	13	19
2	2	8	14	20
3	3	9	15	21
4	4	10	16	22
5	5	11	17	23
6	6	12	18	24

For information on how to display the unique ID and logid numbers of a logical-log file, refer to "Monitoring Logical-Log Files" on page 29-44.

What Are the Status Flags of Logical-Log Files?

All logical-log files have one of the following three status flags in the first position: Added (A), Free (F), or Used (U). Descriptions of all the individual logical-log status flags follow.

Status Flag	Description
Added (A)	A logical-log file has an *added* status when the logical-log file is newly added. It does not become available for use until you complete a level-0 backup of the root dbspace.
Free (F)	A logical-log file is *free* when it is available for use. A logical-log file is freed after it is backed up, all transactions within the log file are closed, and the latest record of a checkpoint is in a subsequent log.
Used (U)	A logical-log file is *used* when it is still needed by the database server for recovery (rollback of a transaction or finding the last checkpoint record).

(1 of 2)

Status Flag	Description
Backed-Up (B)	A log file has a *backed-up* status after the log file has been backed up.
Current (C)	A log file has a *current* status if the database server is currently filling the log file.
Last (L)	A log file has a status of *last* if the log file contains the most recent checkpoint record in the logical log. This file and subsequent files cannot be freed until the database server writes a new checkpoint record to a different logical-log file.

(2 of 2)

Figure 18-4 shows the possible log-status flag combinations.

Figure 18-4
Logical-Log Status Flags

Status Flag	Status of Logical-Log File
A------	Log has been added since the last level-0 dbspace backup. Not available for use.
F------	Log is free. Available for use.
U	Log has been used but not backed up.
U-B----	Log is backed up but still needed for recovery.
U-B---L	Log is backed up but still needed for recovery. Contains last checkpoint record.
U---C	Log is the current log file.
U---C-L	Log is the current log file. It contains the last checkpoint record.

Tip: *A logical-log file has a status flag of F only if the system has been reinitialized. A logical log with just the two status flags U-B (and not L) is available only if it is not spanned by an active transaction.*

To find out the status of a logical-log file, use the methods explained in "Monitoring Logical-Log Files" on page 29-44.

Point-In-Time Recovery

The process of restoring logical logs to a particular point in time is performed after you perform a physical (cold) restore, and it takes effect during a logical restore.

If you use ON-Bar as your backup and restore tool, see your *Backup and Restore Guide* for additional information about restoring logical logs to a particular point in time.

UNIX

If you use ON-Archive as your backup and restore tool, see your *Archive and Backup Guide* for additional information about restoring logical logs to a particular point in time. ♦

Why Do Logical-Log Files Need to Be Backed Up?

The process of copying a logical-log file to tape 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 tape 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.

Use ON-Bar, ON-Archive, or **ontape** to perform a logical-log file backup, depending on which of these tools you use as your backup and restore tool.

Logical-log-file backups can be initiated implicitly as part of continuous logging, or explicitly by the database server administrator or operator. If you use ON-Bar as your backup and restore tool, see your *Backup and Restore Guide* for more information. If you use ON-Archive or **ontape** as your backup and restore tool, see your *Archive and Backup Guide*.

When Are Logical-Log Files Freed?

If you back up a logical-log file, that file is not necessarily free to receive new log records. The following three criteria must be satisfied before the database server frees a logical-log file for reuse:

- The log file is backed up to tape.
- All records within the log file are associated with closed transactions.
- The log file does not contain the most-recent checkpoint record.

When Does the Database Server Attempt to Free a Log File?

The database server attempts to free logical-log files under the following conditions:

- When the database server first writes to a new logical-log file, it attempts to free the previous log.
- Each time that the database server commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began.

The attempt succeeds only if the three criteria listed in the preceding section ("When Are Logical-Log Files Freed?") are met.

What Happens 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 log file.

The logical-log file might be in use for either of the following two reasons:

- The file is not backed up.

  If the log file is not backed up, processing resumes when you perform the backup. If you are using **ontape** to back up logical-log files, you can back up this log file as you would any other log file.

  UNIX

  If you are using ON-Archive to back up logical-log files, you cannot use **onarchive** to back up this log file. The **onarchive** command must be able to access the **sysmaster** database, which it cannot do because processing is suspended. Instead, you must use the **ondatartr** utility to back up the logical-log files in this situation. For more information, see your *Archive and Backup Guide*. ♦

- The file contains an open transaction.

  The open transaction is the long transaction discussed in "Avoiding Long Transactions" on page 18-18. In this situation, you have to recover the database server data from dbspace backup tapes in a full-system restore.

A situation whereby the database server must suspend processing because the next log file contains the last checkpoint never occurs.The database server always forces a checkpoint when it enters the last available log, if the previous checkpoint record 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.

logid	Logical-Log File Status
1	U-B----
2	U---C--
3	F
4	U-B---L

Avoiding Long Transactions

A *long transaction* is a transaction that starts in one logical-log file and is not committed when the database server needs to reuse that same logical-log file. In other words, a long transaction spans more than the total space allocated to the logical log.

Because the database server cannot free a logical-log file until all records within the file are associated with closed transactions, the long transaction prevents the first logical-log file from becoming free and available for reuse.

To prevent long transactions from developing, take the following precautions:

- Ensure that the logical-log file does not fill too fast.
- Ensure that transactions do not remain open too long.
- Set high-water marks to have the database server automatically slow down processing when a long transaction is developing.

The subsequent sections explain these steps.

Factors That Influence the Rate at Which Logical-Log Files Fill

Several factors influence how fast the logical log fills. It is difficult to know exactly which factor is the most important for a given instance of the database server, so you need to use your own judgment to estimate how quickly your logical log fills and how to prevent long-transaction conditions:

- Size of the logical log

  A smaller logical log fills faster than a larger logical log. If you need to make the logical log larger, you can add another logical-log file, explained in "Adding a Logical-Log File" on page 19-4.

- Number of logical-log records

  The more logical-log records written to the logical log, the faster it fills. If databases managed by your database server use transaction logging, transactions against those databases fill the logical log faster than transactions against databases without transaction logging.

- Type of log buffering

  As explained in "Unbuffered Transaction Logging" on page 16-9, databases that use unbuffered transaction logging fill the logical log faster than databases that use buffered transaction logging.

- 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. Also, updates can use up to twice as much space as inserts or deletes because they might contain both before-images and after-images. Inserts store only the after-image, and deletes store only the before-image.

- Frequency of rollbacks

  The frequency of rollbacks affects the rate at which the logical log fills. More rollbacks fill the logical log faster. The rollbacks themselves require logical-log file space although the rollback records are small. In addition, rollbacks increase the activity in the logical log.

Factors That Prevent Closure of Transactions

Several factors influence when transactions close. Be aware of these factors so that you can prevent long-transaction problems:

- Transaction duration

  The duration of a transaction might be beyond your control. For example, a client that does not write many logical-log records might cause a long transaction if the users permit transactions to remain open for long periods of time. (For example, a user who is running an interactive application might leave a terminal to go to lunch part of the way through a transaction.)

  The larger the logical-log space, the longer a transaction can remain open without a long-transaction condition developing. However, a large logical log by itself does not ensure that long transactions do not develop. Application designers should consider the transaction-duration issue, and users should be aware that leaving transactions open can be detrimental.

- High CPU and logical-log activity

  The amount of CPU activity can affect the ability of the database server to complete the transaction. Repeated writes to the logical-log file increase the amount of CPU time that the database server needs to complete the transaction. Increased logical-log activity can imply increased contention of logical-log locks and latches as well.

Setting High-Water Marks

The database server alters processing at two critical points to manage the long-transaction condition. To tune both points, you can set values in the ONCONFIG file.

The first critical point is the *long-transaction high-water mark* described in "LTXHWM" on page 33-51. When the logical log reaches the long-transaction high-water mark, the database server recognizes that a long transaction exists and begins searching for an open transaction in the oldest, used (but not freed) logical-log file. If a long transaction is found, the database server directs the thread to begin to roll back the transaction. More than one transaction can be rolled back if more than one long transaction exists.

The transaction rollback itself generates logical-log records, however, and as other processes continue writing to the logical-log file, the logical log continues to fill.

The second critical point is the *exclusive-access, long-transaction high-water mark* described in "LTXEHWM" on page 33-50. When the logical log reaches the exclusive-access, long-transaction high-water mark, the database server dramatically reduces log-record generation. Most threads are denied access to the logical log. Only threads that are currently rolling back transactions (including the long transaction) 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.

If the long transaction(s) cannot be rolled back before the logical log fills, the database server shuts down. If this situation occurs, you must perform a data restore. During the data restore, you must *not* roll forward the last logical-log file. Doing so re-creates the problem by filling the logical log again.

The default values for the configuration parameters LTXHWM and LTXEHWM are 50 and 60, respectively. These values eliminate any risk of a long transaction having too little log space in which to roll back. The database server initialization emits a warning if your ONCONFIG file contains values greater than 50 and 60 for these parameters. To overcome these warnings, reduce your parameters to conform. If your logspace is finely tuned such that your LTXHWM percentage represents precisely what your longest transaction requires, you will need to add an amount to your log space equal to the difference between your current LTXHWM value and the recommended value of 50.

What Are the Logical-Log Administration Tasks Required for Blobspaces?

Applications that use blobspaces require the following logical-log administration tasks:

- Switching log files to activate blobspaces
- Switching log files to activate new blobspace chunks
- Backing up log files to free deleted blobspace pages

The following sections explain these tasks.

Switching Logical-Log Files to Activate Blobspaces

You must switch to the next logical-log file after you create a blobspace if you intend to insert blobs in the blobspace right away. The database server requires that the statement that creates a blobspace and the statements that insert blobs into that blobspace appear in separate logical-log files. This requirement is independent of the logging status of the database.

For instructions on switching to the next log file, see "Switching to the Next Logical-Log File" on page 19-15.

Switching Logical-Log Files to Activate New Blobspace Chunks

You must switch to the next logical-log file after you add a new chunk to an existing blobspace if you intend to insert blobs in the blobspace that will use the new chunk. The database server requires that the statement that creates a chunk in a blobspace and the statements that insert blobs into that blobspace appear in separate logical-log files. This requirement is independent of the logging status of the database.

For instructions on switching to the next log file, see "Switching to the Next Logical-Log File" on page 19-15.

Backing Up Logical-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 only free 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.

The following sections explain the reasons for this functionality.

Why Do You Have to Back Up Logical-Log Files to Free Blobpages?

Blobs stored in blobspaces generally require much greater amounts of disk space than other data types. For this reason, if an application inserts a row that contains TEXT or BYTE data, the database server does not write the actual column data to the logical-log file. The database server writes only the blobspace overhead pages (which include the free-map page and the bit-map page) to the logical log.

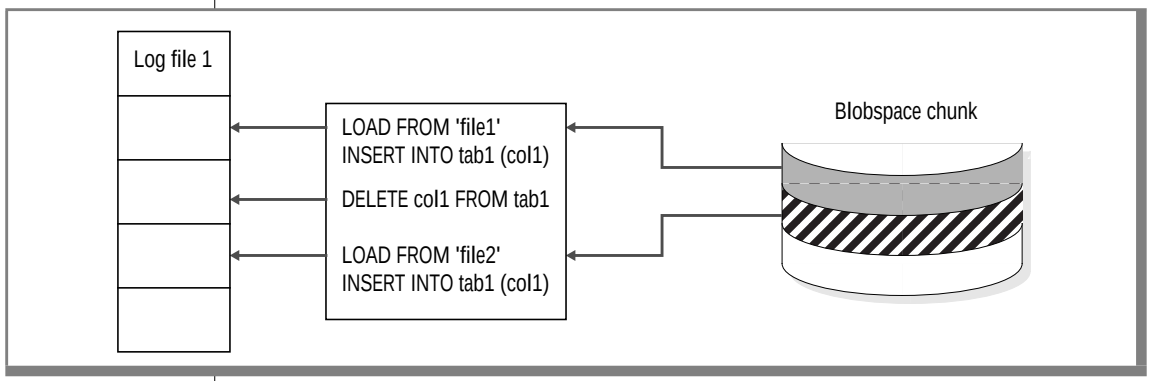
The free-map page contains an entry for each blobpage in the blobspace chunk. Each entry contains the following information:

- A flag that indicates whether the page is used or free
- The unique ID of the logical-log file that was current when the page was written to
- The associated tblspace number of the data stored on the page

For more information on the free-map page, see "Blobspace Page Types" on page 38-64.

When an application deletes a row that contains TEXT or BYTE data, the database server marks each blobpage that contained part of the deleted data as FREE in the free-map overhead page. The database server writes a log record to the logical log to record the changes to the free-map page. If the database server has not yet backed up the logical-log file that contains the transaction that inserted the TEXT or BYTE data, the database server does not write over the deleted blobpages on subsequent inserts of TEXT or BYTE data. Figure 18-5 illustrates this scenario.

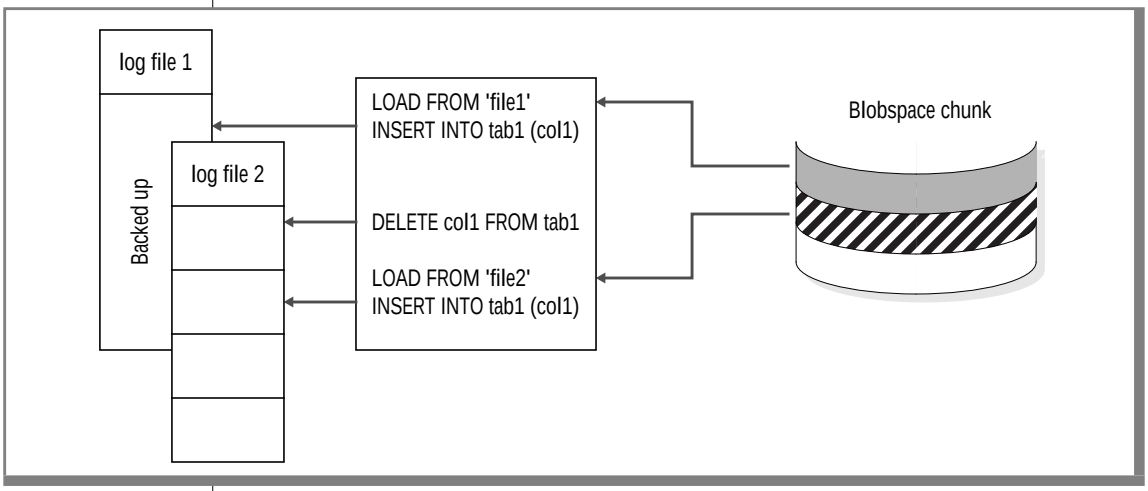
Figure 18-5
No Write to Deleted Blobpages If Logical-Log File Is Not Backed Up



The database server does not write over the deleted blobpages in the scenario shown in Figure 18-5 because it might need to perform a fast recovery and roll forward the transaction that contains the original insert. If the original data was overwritten, the database server would have no way of reproducing the transaction.

When you back up the logical-log file that contains the original insert, however, the database server copies the actual TEXT or BYTE data to tape. After the data is saved to tape and available for a recovery, the database server can write over the deleted blobpages, as shown in Figure 18-6.

Figure 18-6
Write to Deleted Blobpages If Logical-Log File Is Backed Up



What Is the Logging Process?

This section describes in detail the logging process for both dbspace and blobspace logging. 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. Read the data page from disk to the shared-memory page buffer.
2. Copy the unchanged page to the physical-log buffer.
3. Write the new data into the page buffer, and create a logical-log record of the transaction, if needed.
4. Flush physical-log buffer to the physical log on disk.
5. Flush the page buffer, and write it back to disk.
6. Flush the logical-log buffer to a logical-log file on disk.

Read Page into Shared-Memory Buffer Pool

In general, an insert or an update begins when a thread requests a row. The database server identifies the page on which the row resides and attempts to locate the page in the shared-memory buffer pool. If the page is not already in shared memory, the database server reads the page from disk. "How a Database Server Thread Accesses a Buffer Page" on page 11-43 explains this process in more detail.

Copy the Page Buffer into the Physical-Log Buffer

Before the database server modifies a dbspace data page, it stores a copy of the unchanged page in the physical-log page buffer. The database server eventually flushes the physical-log page buffer that contains this *before-image* to the physical log on disk. The before-image of the page plays a critical role in fast recovery. Until the database server performs a new checkpoint, subsequent modifications to the same page do not require another before-image to be stored in the physical-log buffer. For more information, refer to "Flushing the Physical-Log Buffer" on page 11-47.

The database server knows if a page is already in the physical log. If the time stamp on the page is more recent than the time stamp for the last checkpoint, the page has been changed since the checkpoint and is therefore already in the physical log.

Read Data into Buffer and Create Logical-Log Record

The thread that performs the modifications receives data from the application. After the database server stores a copy of the unchanged data page in the physical-log buffer, the thread writes the new data to the page buffer and writes records necessary to roll back or re-create the operation to the logical-log buffer. For more information, refer to "When the Logical-Log Buffer Becomes Full" on page 11-54.

Flush Physical-Log Buffer to the Physical Log

The database server must flush the physical-log buffer before it flushes the data buffer. Flushing the physical-log buffer ensures that a copy of the unchanged page is available until the changed page is written to the physical log. For more information, refer to "Flushing the Physical-Log Buffer" on page 11-47.

Flush Page Buffer

At some point after the database server flushes the physical-log buffer, the database server flushes the data buffer and writes the modified data page to disk. This action occurs at the next checkpoint, or when a page cleaner determines that the page should be written to disk. The database server does not flush the data buffer as the transaction is committed. For more information, see "How the Database Server Flushes Data to Disk" on page 11-46.

Flush Logical-Log Buffer

To flush the logical-log buffer, the database server writes the logical-log records to the current logical-log file on disk. For more information, see "Flushing the Logical-Log Buffer" on page 11-53.

A logical-log file cannot become free, and available for reuse, until all transactions represented in the log file are completed and the log file is backed up to tape. This requirement ensures that all open transactions can be rolled back, if required.

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 bit-map pages) are the only blobspace data that reaches the logical log. By logging these overhead pages, the logical-log file records track blobpage allocation and deallocation (when TEXT or BYTE data is deleted from blobpages), but not the actual TEXT or BYTE data. Blobspace data is recorded in the logical log only when a log file is backed up to tape.

Blobspace logging occurs in the following three steps:

1. Blobspace data flows from the network, through temporary buffers in the database server process memory space, and is written directly to disk. If the TEXT or BYTE object requires more than one blobpage, the database server creates links and pointers as needed.
2. A record of the operation (insert, update, or delete) is written to the logical-log buffer, if the database uses logging. The TEXT or BYTE data is not included in the record (but the information about where the TEXT or BYTE data is placed is included by way of the overhead pages).
3. When a logical-log backup begins, the database server uses the logical-log ID number stored in the blobspace free-map page to determine which blobpages to copy to tape. If you use ON-Bar as your backup and restore tool, see your *Backup and Restore Guide* for more details on logical-log backups. If you use ON-Archive as your backup and restore tool, see your *Archive and Backup Guide*.

Chapter

19

Managing Logical-Log Files

Adding a Logical-Log File 19-4
Using ON-Monitor to Add a Log File 19-5
Using onparams to Add a Log File 19-5
Adding a Log File with a New Size 19-5

Dropping a Logical-Log File. 19-6
Using ON-Monitor to Drop a Logical-Log File 19-7
Using onparams to Drop a Logical-Log File 19-7

Moving a Logical-Log File to Another Dbspace 19-8
An Example of Moving Logical-Log Files. 19-8

Changing the Size of Logical-Log Files 19-9

Changing Logical-Log Configuration Parameters 19-10
Changing LOGSIZE or LOGFILES 19-10
Using ON-Monitor to Change LOGSIZE or LOGFILES. . . . 19-11
Using a Text Editor to Change LOGSIZE or LOGFILES. . . . 19-11
Changing LOGSMAX, LTXHWM, or LTXEHWM 19-12
Changing LOGSMAX, LTXHWM, or LTXEHWM with ON-Monitor 19-12
Editing the ONCONFIG File to Change LOGSMAX, LTXHWM, or LTXEHWM 19-13

Freeing a Logical-Log File 19-13
Freeing a Log File with Status A 19-13
Freeing a Log File with Status U 19-14
Freeing a Log File with Status U-B 19-14
Freeing a Log File with Status U-C or U-C-L. 19-14
Freeing a Log File with Status U-B-L 19-15

Switching to the Next Logical-Log File 19-15

This chapter contains information on managing the database server logical-log files with command line utilities. You must manage logical-log files even if none of your databases uses transaction logging.

The chapter covers the following tasks:

- Adding a logical-log file
- Dropping a logical-log file
- Moving a logical-log file
- Changing the size of a logical-log file
- Changing the logical-log configuration parameters
- Freeing a logical-log file
- Switching to the next logical-log file

Several of the tasks described in this chapter can be accomplished through INFORMIX-Enterprise Command Center (IECC). For instructions on backing up logical-log files, enabling logical logging, and disabling logical logging through IECC, see the *INFORMIX-Enterprise Command Center User Guide*.

For background information regarding the logical log, refer to Chapter 18, "What Is the Logical Log?"

For instructions on backing up logical-log files with ON-Bar, refer to your *Backup and Restore Guide*. For instructions on backing up logical-log files with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

Adding a Logical-Log File

You might add a log file for the following reasons:

- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- As part of moving logical files to a different dbspace

You add log files one at a time. You cannot add a log file during a dbspace backup (quiescent or on-line). The database server must be in quiescent mode to add a logical-log file.

A newly added log file does not become available until you create a level-0 dbspace backup of the root dbspace. This requirement ensures that the dbspace backup copy of the reserved pages contains information about the current number of logical-log files. Use your backup tool to create the level-0 dbspace backup.

You can use either of the following utilities to add the log file:

- **onparams**
- ON-Monitor

UNIX

If you use ON-Monitor, the size of the new log file is always the value specified by LOGSIZE. ♦

You must use the **onparams** utility to add a new log file with a different size than LOGSIZE.

Verify that you will not exceed the maximum number of logical-log files allowed in your configuration, specified as LOGSMAX. If you need to, you can increase LOGSMAX (as described in "Changing LOGSMAX, LTXHWM, or LTXEHWM" on page 19-12) and reinitialize shared memory for the change to take effect.

You must be logged in as either **informix** or **root** to make this change.

UNIX

Using ON-Monitor to Add a Log File

Bring the database server to quiescent mode. **Select Parameters→ Add-Log** option to add a logical-log file.

Enter the name of the dbspace where the new logical-log file will reside in the field labelled **Dbspace Name**. The size of the log file automatically appears in the **Logical Log Size** field.

After you add the log file, the status of the new log file is A. The newly added log file becomes available after you create a level-0 dbspace backup of the root dbspace. For information on creating a level-0 dbspace backup with ON-Bar, refer to the *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the *Archive and Backup Guide*.

Using onparams to Add a Log File

The following command adds a logical-log file to the dbspace called **logspace**, using the log-file size specified by the LOGSIZE configuration parameter:

```
onparams -a -d logspace
```

The status of the new log file is A. The newly added log file becomes available after you create a level-0 dbspace backup of the root dbspace. For information on creating a level-0 dbspace backup with ON-Bar, refer to your *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

For reference information on using **onparams** to add a logical-log file, see "Add a Logical-Log File" on page 35-44.

Adding a Log File with a New Size

To add a logical-log file with a size different from that specified by LOGSIZE (in this case, 250 kilobytes) to a dbspace called **logspace**, execute the following command:

```
onparams -a -d logspace -s 250
```

Adding a log file of a new size does not change the value of LOGSIZE.

The status of the new log file is A. The newly added log file becomes available after you create a level-0 dbspace backup of the root dbspace. For information on creating a level-0 dbspace backup with ON-Bar, refer to your *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*

For reference information on using **onparams** to add a logical-log file, see "Add a Logical-Log File" on page 35-44.

Dropping a Logical-Log File

You can drop a log to increase the amount of the disk space available within a dbspace.

The database server requires a minimum of three logical-log files at all times. (Log files that are newly added and have status A do not count toward this minimum of three.) You cannot drop a log if your logical log is composed of only three log files.

You drop log files one at a time. After your configuration reflects the desired number of logical-log files, create a level-0 dbspace backup of the root dbspace. This action ensures that the dbspace backup copy of the reserved pages contains information about the current number of logical-log files. This information prevents the database server from attempting to use the dropped log files during a restore.

You can only drop a log file that has a status of Free (F) or newly Added (A).

You must know the logid number of each logical log that you intend to drop. For information on obtaining a display of the logical-log files and logid numbers, see "Monitoring Logical-Log Files" on page 29-44.

To make this change, you must log in as either **informix** or **root**, and the database server must be in quiescent mode.

UNIX

Using ON-Monitor to Drop a Logical-Log File

Select **Parameters→Drop-Log** to drop a logical-log file. Use the arrow keys to select the log you want to drop and press CTRL-B or F3. You are asked to confirm your choice.

Create a level-0 dbspace backup of the root dbspace after your configuration reflects the desired number of logical-log files. For information on creating a level-0 dbspace backup with ON-Bar, refer to your *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

Using onparams to Drop a Logical-Log File

Execute the following command to drop a logical-log file whose logid number is 21:

```
onparams -d -l 21
```

Create a level-0 dbspace backup of the root dbspace after your configuration reflects the number of logical-log files you want. For information on creating a level-0 dbspace backup with ON-Bar, refer to your *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

See "Drop a Logical-Log File" on page 35-45 for reference information on using **onparams** to drop a logical-log file.

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 "Where Should Logical-Log Files Be Located?" on page 18-12. To find out the location of logical-log files, see "Monitoring Logical-Log Files" on page 29-44.

Changing the location of the logical-log files is actually a combination of two simpler actions:

- Dropping logical-log files from their current dbspace
- Adding the logical-log files to their new dbspace

Although moving the logical-log files is not difficult, it can be time-consuming because you must create two separate level-0 dbspace backups of the root dbspace as part of the procedure. For an example of the procedure, see "An Example of Moving Logical-Log Files" on page 19-8.

An Example of Moving Logical-Log Files

The database server must be in quiescent mode to make these changes.

The following procedure provides an example of how to move six logical-log files from the **root** dbspace to another dbspace, **dbspace_1**:

1. Free all log files except the current log file.

   See "Freeing a Logical-Log File" on page 19-13.

2. Verify that the value of LOGSMAX is greater than or equal to the number of log files after the move plus 3.

   In this case, the value of LOGSMAX must be greater than or equal to 9. Change the value of LOGSMAX, if necessary. See "Changing LOGSMAX, LTXHWM, or LTXEHWM" on page 19-12.

3. Drop all but three of the logical-log files.

   You cannot drop the current log file. If you have only three logical-log files in the root dbspace, skip this step.

   See "Dropping a Logical-Log File" on page 19-6.

4. Add the new log files to the different dbspace.

   In this case, add six new log files to **dbspace_1**. See "Adding a Logical-Log File" on page 19-4.

5. Create a level-0 dbspace backup of the root dbspace to make the new log files available to the database server.

   For information on creating a level-0 dbspace backup with ON-Bar, refer to your *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape,** refer to your *Archive and Backup Guide*.

6. Switch the logical-log files to start a new current log file.

   See "Switching to the Next Logical-Log File" on page 19-15.

7. Back up the former *current log file* to free it.

   For information on backing up logical-log files with ON-Bar, refer to your *Backup and Restore Guide*. For information on backing up logical-log files with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

8. Drop the three log files that remain in the root dbspace.

   For information on dropping a logical-log file, refer to "Dropping a Logical-Log File" on page 19-6.

Changing the Size of Logical-Log Files

You can change the size of logical-log files in two ways:

- Use **onparams** to add a new log file of a different size.

  This change has no effect on LOGSIZE. The log files that you add are available after the next level-0 dbspace backup of the root dbspace (instead of after reinitializing disk space). See "Adding a Log File with a New Size" on page 19-5.

- Change the LOGSIZE configuration parameter.

  Changing LOGSIZE changes the default size for all subsequent logical-log files added but is time-consuming because it requires that you reinitialize disk space to see the change. See "Changing LOGSIZE or LOGFILES" on page 19-10.

Changing Logical-Log Configuration Parameters

The following configuration parameters affect the logical-log file and how the database server works with it:

- LOGSIZE (described on page 33-43)
- LOGFILES (described on page 33-42)
- LOGSMAX (described on page 33-44)
- LTXHWM (described on page 33-51)
- LTXEHWM (described on page 33-50)

The following sections explain the procedure for changing each of these parameters.

Changing LOGSIZE or LOGFILES

UNIX

You must be logged in as **root** or **informix** to change these configuration parameters. You can change LOGSIZE or LOGFILES in two ways:

- Using ON-Monitor ♦
- Using a text editor

In each case, the changes to LOGSIZE and LOGFILES do not take effect until you reinitialize the disk. To retain your existing data when you reinitialize the disk, you must unload the data beforehand and reload it once the disk is initialized. This process makes changing these parameters relatively difficult.

If you want to increase the number of log files, you can add log files more easily one at a time, as discussed in "Adding a Logical-Log File" on page 19-4. Similarly, if you want to change the size of the log files, you might find it easier to add new log files of the desired size and then drop the old ones.

UNIX

Using ON-Monitor to Change LOGSIZE or LOGFILES

You can use ON-Monitor to change the value of LOGSIZE or LOGFILES.

To change these values

1. Unload all the database server data.

   You cannot rely on dbspace backup tapes to unload and restore the data because a restore returns the parameters to their previous value.
2. Select **Parameters→Initialize** option to reinitialize disk space.
3. Change the value of LOGSIZE or LOGFILES.

   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 about the **onload** utility and about unloading data, see the *Informix Migration Guide*.

Using a Text Editor to Change LOGSIZE or LOGFILES

You can change the value of LOGSIZE or LOGFILES by using an editor to edit the ONCONFIG file.

To change these values

1. Change the value of LOGSIZE or LOGFILES.
2. Unload all the database server data.

   You cannot rely on dbspace backup tapes to unload and restore the data because a restore returns the parameters to their previous value.
3. Use **oninit** to reinitialize disk space.
4. Re-create all databases and tables.
5. Reload all the database server data.

For information about the **onload** utility and about moving tables and data, see the *Informix Migration Guide*.

Changing LOGSMAX, LTXHWM, or LTXEHWM

You can change the value of LOGSMAX, LTXHWM, or LTXEHWM with the following methods:

UNIX

- Using ON-Monitor ♦
- Editing the ONCONFIG file

Each of these methods is explained in the following sections. Changes to these configuration parameters take effect when you reinitialize shared memory.

You must be logged in as **root** or **informix** to change these configuration parameters.

UNIX

Changing LOGSMAX, LTXHWM, or LTXEHWM with ON-Monitor

You can use ON-Monitor to change LOGSMAX, LTXHWM, or LTXEHWM while the database server is in on-line mode.

Select **Parameters→Shared-Memory** option to change one or more of the values. ON-Monitor displays the current values, and you can change the values as follows.

To change this value	Change this ON-Monitor field
LOGSMAX	MAX # of Logical Logs
LTXHWM	Long TX HWM
LTXEHWM	Long TX HWM Exclusive

Reinitialize shared memory for the change or changes to take effect. See "Adding a Segment to the Virtual Portion of Shared Memory" on page 12-15.

Editing the ONCONFIG File to Change LOGSMAX, LTXHWM, or LTXEHWM

You can change the value of LOGSMAX, LTXHWM, or LTXEHWM by using a text editor to edit the ONCONFIG file.

Change the value of the parameter that you wish to change.

Reinitialize shared memory for the change to take effect. See "Adding a Segment to the Virtual Portion of Shared Memory" on page 12-15.

Freeing a Logical-Log File

For a description of what constitutes a free logical-log file, see "What Are the Status Flags of Logical-Log Files?" on page 18-13.

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 "Monitoring Logical-Log Files" on page 29-44.

Freeing a Log File with Status A

If a log file is newly added (status A), create a level-0 dbspace backup of the root dbspace to activate the log file and make it available for use. For information on creating a level-0 dbspace backup with ON-Bar, refer to your *Backup and Restore Guide*. For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

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 dbspace backup and backup tool that you usually use. For information on backing up logical-log files with ON-Bar, refer to your *Backup and Restore Guide*. For information on backing up logical-log files with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

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 "Freeing a Log File with Status U-B-L" on page 19-15.

Freeing a Log File with Status U-B

If a log file is backed up but still in use (status U-B), some transactions in the log file are still under way. If you do not want to wait until the transactions complete, take the database server to quiescent mode. See "Immediately from On-Line to Quiescent" on page 7-7. Any active transactions are rolled back.

Freeing a Log File with Status U-C or U-C-L

If you want to free the current log file (status C), follow these steps:

1. Execute the following command:

   ```
   % onmode -l
   ```

   (Be sure to type a lowercase L on the command line, not a number 1.) This command switches the current log file to the next available log file.

2. Back up the original log file with the dbspace backup and backup tool that you usually use. For information on backing up logical-log files with ON-Bar, refer to your *Backup and Restore Guide*. For information on backing up logical-log files with ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

   After all full log files are backed up, you are prompted to switch to the next available logical-log file and back up the new current log file. You do not need to do this because you just switched to this log file.

After you follow these steps, if the log file now has status U-B or U-B-L, refer to "Freeing a Log File with Status U-B" on page 19-14 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 to tape 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

You can also use ON-Monitor to force a checkpoint. To force a checkpoint with ON-Monitor, select the Force-Ckpt option. ♦

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 activate new blobspaces
- To activate new blobspace chunks
- To back up the current log

The database server can be in on-line 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.)

Chapter

20

What Is Physical Logging?

What Is Physical Logging? 20-3
What Is the Purpose of Physical Logging? 20-4
Fast Recovery Uses Physically Logged Pages 20-4
Backup Uses Physically Logged Pages 20-4
What Database Server Activity Is Physically Logged? 20-4
Are Blobs Physically Logged? 20-5

How Big Should the Physical Log Be? 20-5
Can the Physical Log Become Full? 20-7

Where Is the Physical Log Located? 20-8

Details of Physical Logging 20-9
Page Is Read into the Shared-Memory Buffer Pool. 20-9
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer 20-10
Change Is Reflected in the Data Buffer 20-10
Physical-Log Buffer Is Flushed to the Physical Log 20-10
Page Buffer Is Flushed 20-11
When a Checkpoint Occurs 20-11
How the Physical Log Is Emptied 20-11

This chapter defines the terms and explains the concepts that you need to know to perform effectively the tasks described in Chapter 21, "Managing the Physical Log." The chapter covers the following topics:

- What physical logging is and what purposes it serves
- What the physical log is, and some guidelines for it size and location
- Details of the physical-logging process

What Is 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. Before the database server modifies a page in the shared-memory buffer pool, it stores an unmodified copy of the page (called a *before-image*) in the physical-log buffer in shared memory.

The *physical log* is a set of contiguous disk pages where the database server stores before-images.

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. (See "Checkpoints" on page 11-57.) Once a checkpoint occurs, the database server empties the physical log (except in the special circumstances explained in "Can the Physical Log Become Full?" on page 20-7).

What Is the Purpose of Physical Logging?

This seemingly odd activity of storing copies of pages before they are changed ensures that 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. These snapshots are potentially used in two activities: fast recovery and the database server backup.

Fast Recovery Uses Physically Logged Pages

After a failure, the database server uses the before-images in the physical log to restore all pages on the disk to their state at the last checkpoint. When the before-image pages are combined with the logical-log records stored since the checkpoint, the database server can return all data to physical and logical consistency, up to the point of the most-recently completed transaction. Chapter 22, "What Is Fast Recovery?" explains this procedure in more detail.

Backup Uses Physically Logged Pages

When you perform an on-line dbspace backup, the database server checks disk pages to see which should be backed up. As part of this process, the database server periodically reads the pages in the physical log. If the database server finds pages in the physical log that meet the backup criterion, they are copied to the backup tape. For more detailed information, see your *Archive and Backup Guide* if you use ON-Archive as your backup and restore tool or your *Backup and Restore Guide* if you use ON-Bar.

What Database Server Activity Is Physically Logged?

All dbspace page modifications except the following ones are physically logged:

- Pages that do not have a valid database server address

  This situation usually occurs when the page was used by some other database server or a table that was dropped.
- Pages that the database server has not allocated and that are located in a dbspace where no table has been dropped since the last checkpoint

In case of multiple modifications before the next checkpoint, only one before-image is logged in the physical log (the first before-image).

Storing all before-images of page modifications in the physical log might seem excessive. But the database server stores the before-images in the physical log only until the next checkpoint. To control the amount of data that the database server logs, you can tune the checkpoint interval configuration parameter CKPTINTVL.

Are Blobs Physically Logged?

The database server pages in the physical log can be any database server page except a blobspace blobpage. Even overhead pages (such as chunk free-list pages, blobspace free-map pages, and blobspace bit-map pages to the free-map pages) are copied to the physical log before data on the page is modified and flushed to disk, but blobspace blobpages are not. For further information about blobspace logging, see "Are Blobs Logged?" on page 16-7.

How Big Should the Physical Log Be?

When you consider how large to make your physical log, you can begin by using the following formula to calculate an approximate size:

```
Physical Log Size = userthreads * max_log_pages_per_critical_section * 4
```

This formula is based on how much physical logging space the database server needs in a worst-case scenario. This scenario takes place when a checkpoint occurs because the log becomes 75 percent full. If all the update threads are in a critical section (see "Critical Sections" on page 11-56) and perform physical logging of the maximum number of pages in their critical section, the database server must fit this logging into the final 25 percent of the physical log to prevent a physical-log overflow.

To obtain an estimate for the number of *userthreads*, execute **onstat -u** during peak processing. The last line of **onstat -u** contains the maximum number of concurrent user threads. Substitute this number for *userthreads* in the formula.

The maximum number of pages (*max_log_pages_per_critical_section*) that the database server can physically log in a critical section is five. The number four in the formula is necessary because the following part of the formula represents only 25 percent of the physical log:

```
userthreads * max_log_pages_per_critical_section
```

The exception to this rule occurs if you are using tblspace blobs in a database without logging. Here, substitute the size of the most-frequently occurring blob in the dbspace for the maximum log pages per critical section.

Also consider the following issues:

- How much updating of data does the database server perform?

  Operations that do not perform updates do not generate before-images. If the applications that use your database server do not perform much updating, you might not need a very big physical log. If the size of your database is fixed, but you frequently update the data, a lot of physical logging occurs. If the size of the database is growing, but applications rarely update the data, not much physical logging occurs.

  The database server writes the before-image of only the first update made to a page. Thus, if your application repeatedly updates the same pages, you need a smaller physical log than if your application performs a lot of updating but seldom updates the same page.

- How frequently do checkpoints occur?

  Because the physical log is emptied after each checkpoint, the physical log only needs to be large enough to hold before-images from changes between checkpoints. If your physical log frequently approaches full, you might consider decreasing the checkpoint interval, CKPTINTVL, so that checkpoints occur more frequently. However, decreasing the checkpoint interval beyond a certain point has an impact on performance.

  If you plan to increase the checkpoint interval, or if you anticipate increased activity, you will probably want to increase the size of the physical log.

The size of the physical log is specified by the ONCONFIG parameter PHYSFILE. (See "PHYSFILE" on page 33-71.)

Can the Physical Log Become Full?

Because a checkpoint is initiated that logically empties the physical log when it becomes 75 percent full, it is unlikely that the log would become 100 percent full before the checkpoint completes. To assure further that the physical log does not become full during a checkpoint, take the following actions:

- Configure the database server according to the sizing guidelines for the physical log and the logical-log files.
- Fine-tune the size of the physical log by monitoring it during production activity.

However, the physical log could still become full as described in the following paragraphs.

Under normal processing, once a checkpoint is requested, and the checkpoint begins, all threads are prevented from entering critical sections of code. (See "Critical Sections" on page 11-56.) However, threads *currently* in critical sections can continue processing. The physical log can become full if many threads in critical sections are processing work *and* if the space that remains in the physical log is very small. The many writes performed as threads complete their critical section processing could conceivably cause the physical log to become full.

Consider the following example. When the database server processes tblspace blobs stored in a database created with transaction logging, each portion of the blob 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 the database was created without logging, the database server must carry out all operations on the tblspace blob in one critical section. If the blob is large, and the physical log small, this scenario can cause the physical log 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 this message in your message log.

This same unlikely scenario could occur during the rollback of a long transaction after the second long-transaction high-water mark, LTXEHWM, is reached. (See "Avoiding Long Transactions" on page 18-18.) After the LTXEHWM is reached, and after all threads have exited critical sections, only the thread that is performing the rollback has access to the physical and logical logs. However, the writes that are performed as threads complete their processing could conceivably fill the physical log during the rollback if the following conditions occur simultaneously:

- Many threads were in critical sections.
- The space remaining in the physical log was very small at the time that the LTXEHWM was reached.

Where Is the Physical Log Located?

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 on a disk that does not contain active tables or the logical-log files.

The physical log is located in the dbspace specified by the ONCONFIG parameter PHYSDBS. (See "PHYSDBS" on page 33-70). Change this parameter only if you decide to move the physical-log file from the root dbspace. (See "Changing the Physical-Log Location and Size" on page 21-3.)

Because of the critical nature of the physical log, Informix recommends that you mirror the dbspace that contains the physical log.

Details of Physical Logging

This section describes the details of physical logging. It is provided to satisfy your curiosity; you do not need to understand the information here in order to manage your physical log.

The database server performs physical logging in the following six steps:

1. Reads the data page from disk to the shared-memory page buffer (if the data page is not there already)
2. Copies the unchanged page to the physical-log buffer
3. Reflects the change in the page buffer after an application modifies data
4. Flushes the physical-log buffer to the physical log on disk
5. Flushes the page buffer and writes it back to disk
6. When a checkpoint occurs, flushes the physical-log buffer to the physical log on disk and empties the physical log

The paragraphs that follow explain each step in detail.

Page Is Read into the Shared-Memory Buffer Pool

When a session requests a row, the database server identifies the page on which the row resides and attempts to locate the page in the database server shared-memory buffer pool. If the page is not already in shared memory, it is read into the resident portion of the database server shared memory from disk.

A Copy of the Page Buffer Is Stored in the Physical-Log Buffer

Before a dbspace data page is modified, a copy of the unchanged page is stored in the physical-log buffer (if the unchanged page is not already stored in the physical-log buffer since the last checkpoint). This copy of the before-image of the page is eventually flushed from the physical-log buffer to the physical log on disk. The before-image of the page plays a critical role in archiving and fast recovery. (Subsequent modifications of the same page before the next checkpoint do not require another before-image to be stored in the physical-log buffer.)

Change Is Reflected in the Data Buffer

The application changes data. The database server reflects these changes in the shared-memory data buffer.

Data from the application is passed to the database server. After a copy of the unchanged data page is stored in the physical-log buffer, the new data is written to the page buffer already acquired.

Physical-Log Buffer Is Flushed to the Physical Log

The database server will probably flush the physical-log buffer before it flushes the data buffer to ensure that a copy of the unchanged page is available until the changed page is copied to disk. The before-image of the page is no longer needed after a checkpoint occurs. (During a checkpoint, all modified pages in shared memory are flushed to disk, providing a consistent point from which to recover in case an uncontrolled shutdown occurs.)

Page Buffer Is Flushed

After the physical-log buffer is flushed, the shared-memory page buffer is flushed to disk (but only as a result of a fixed set of conditions such as a checkpoint), and the data page is written to disk. For conditions that lead to the flushing of the page buffer, see "How the Database Server Achieves Data Consistency" on page 11-56.

When a Checkpoint Occurs

A checkpoint can occur at any point in the physical-logging process. After a checkpoint occurs, the database server is physically consistent. The data on disk reflects the actual changes that the application made since the data pages in shared memory were flushed to disk. The database server empties the physical log logically, allowing current entries to be overwritten.

How the Physical Log Is Emptied

The database server manages the physical log as a circular file, constantly overwriting unneeded data. The checkpoint procedure empties the physical log by resetting a pointer in the physical log that marks the beginning of the next group of required before-images.

Chapter

21

Managing the Physical Log

Changing the Physical-Log Location and Size 21-3
- Why Change Physical-Log Location and Size? 21-4
- Before You Make the Changes. 21-4
- Using ON-Monitor to Change Physical-Log Location or Size 21-5
- Using a Text Editor to Change Physical-Log Location and Size. 21-5
- Using onparams to Change Physical-Log Location or Size 21-6

This chapter describes how to change the configuration parameters associated with the physical log. For background information about the physical log, see Chapter 20, "What Is Physical Logging?"

Changing the Physical-Log Location and Size

UNIX

You can change your physical-log location or size in three ways:

- Using ON-Monitor ♦
- Using a text editor to edit the ONCONFIG file
- Using the **onparams** utility from the command line

Log in as user **informix** or **root** on a UNIX system or as a member of the **Informix-Admin** group on a Windows NT system when you make the changes. The following sections describe each of these methods.

For any of the three methods, to activate the changes to the size or location of the physical log as soon as you make them, reinitialize shared memory. If you use **onparams**, you can reinitialize shared memory in the same step.

Create a level-0 dbspace backup immediately after you reinitialize shared memory. This dbspace backup is critical for database server recovery.

Why Change Physical-Log Location and Size?

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.

For advice on where to place the physical log, see "Where Is the Physical Log Located?" on page 20-8. For advice on sizing the physical log, see"How Big Should the Physical Log Be?" on page 20-5.

Before You Make the Changes

The space allocated for the physical log must be contiguous. If you move the log to a dbspace without adequate contiguous space, or if you increase the log size beyond the available contiguous space, a fatal shared-memory error occurs when you attempt to reinitialize shared memory with the new values. If this error occurs, resize the log, or choose another dbspace with adequate contiguous space and then reinitialize the database server.

You can check if adequate contiguous space is available with the **-pe** option of the **oncheck** utility. For more information, see "Monitoring Chunks" on page 29-53.

To change the physical-log location or size, you must log in as user **root** or **informix** on a UNIX system or as a member of the **Informix-Admin** group on a Windows NT system.

UNIX

Using ON-Monitor to Change Physical-Log Location or Size

Select **Parameters→Physical-Log** to change the size or dbspace location, or both.

The **Physical-log Size** field displays the current size of the log. Enter the new size (in kilobytes) if you want to change the size of the log. The **Dbspace Name** field displays the current location of the physical log. Enter the name of the new dbspace if you want to change the log location.

You are prompted, first, to confirm the changes.

```
Do you really want to shut down?
```

The second prompt checks if you want to shut the database server down.

```
Do you really want to continue?
```

This last message refers to reinitializing shared memory. If you respond `Y`, ON-Monitor reinitializes shared memory, and any changes are implemented immediately. The database server displays messages that the database server is shutting down and then initializing and recovering. If you respond `N`, the values are changed in the configuration file, but the changes do not take effect until you reinitialize shared memory.

After you reinitialize shared memory, create a level-0 dbspace backup immediately to ensure that all recovery mechanisms are available.

Using a Text Editor to Change Physical-Log Location and Size

You can change the value of the following parameters (page numbers point you to instructions) in the ONCONFIG file with your text editor while the database server is in on-line mode:

- PHYSFILE page 33-71
- PHYSDBS page 33-70

The changes do not take effect until you reinitialize shared memory.

After you reinitialize shared memory, create a level-0 dbspace backup immediately to ensure that all recovery mechanisms are available.

Using onparams to Change Physical-Log Location or Size

You can find reference information regarding the **onparams** utility in "onparams: Modify Log-Configuration Parameters" on page 35-43.

To change the size and location of the physical log, execute the following command after you bring the database server to quiescent 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. The command also reinitializes shared memory with the **-y** option so that the change takes effect immediately, as follows:

```
% onparams -p -s 400 -d dbspace6 -y
```

After you reinitialize shared memory, create a level-0 dbspace backup to ensure that all recovery mechanisms are available. If you use ON-Bar as your backup and restore tool, refer to your *Backup and Restore Guide* for information on performing a level-0 dbspace backup. For information on performing a level-0 dbspace backup using ON-Archive or **ontape**, refer to your *Archive and Backup Guide*.

Chapter

22

What Is Fast Recovery?

What Is Fast Recovery? 22-3
When Is Fast Recovery Needed? 22-4
When Does the Database Server Initiate Fast Recovery? 22-4
Fast Recovery and Buffered Logging 22-4
Fast Recovery and No Logging 22-5

Details of Fast Recovery 22-5
Returning to the Last-Checkpoint State 22-6
Finding the Checkpoint Record in the Logical Log. 22-6
Rolling Forward Logical-Log Records 22-7
Rolling Back Incomplete Transactions 22-8

This chapter describes the database server fast-recovery feature. You do not need to take any administrative actions with respect to fast recovery; it is an automatic feature. Read this chapter if you are interested in what fast recovery is and how it works.

This chapter covers the following topics:

- A definition of fast recovery
- The types of failures that fast recovery addresses
- How the database server detects these failures
- The details of how fast recovery works

What Is Fast Recovery?

Fast recovery is an automatic, fault-tolerant feature that the database server executes every time it moves from off-line to quiescent mode or from off-line to on-line mode.

The fast-recovery process checks if, the last time that the database server went off-line, it did so in uncontrolled conditions. If so, fast recovery returns the database server to a state of physical and logical consistency, as described in "Details of Fast Recovery" on page 22-5.

If the fast-recovery process finds that the database server came off-line in a controlled manner, the fast-recovery process terminates, and the database server moves to on-line mode.

When Is Fast Recovery Needed?

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. Such failures are usually caused by system failures. System failures do not damage the database but instead affect transactions that are in progress at the time of the failure.

Fast recovery addresses the following kinds of system failure:

- The database server is processing tasks for more than 40 users.
- Dozens of transactions are on-going.
- Without warning, the operating system fails.

How does the database server bring itself to a consistent state again? What happens to ongoing transactions? The answer to both questions is fast recovery.

When Does the Database Server Initiate Fast Recovery?

Every time the administrator brings the database server to quiescent mode or on-line mode from off-line mode, the database server checks 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 on-line 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 off-line 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 16-9), 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 is unable to 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.)

Fast Recovery and No Logging

For databases 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.

Details of Fast Recovery

Fast recovery returns the database server to a consistent state as part of shared-memory initialization. The consistent state means that all committed transactions are restored, and all uncommitted transactions are rolled back.

Fast recovery is accomplished in the following two stages:

- The database server uses the physical log to return to the most recent point of known *physical consistency*, the most recent checkpoint.
- The database server uses the logical-log files to return to *logical consistency* by rolling forward all committed transactions that occurred after the last checkpoint and rolling back all transactions that were left incomplete.

The two stages can also be expressed as the following four steps. Each step is described in detail in the paragraphs that follow:

1. Use the data in the physical log to return all disk pages to their condition at the time of the most recent checkpoint.
2. Locate the most recent checkpoint record in the logical-log files.
3. Roll forward all logical log records written after the most recent checkpoint record.
4. Roll back transactions that do not have an associated COMMIT record in the logical log.

Returning to the Last-Checkpoint State

To accomplish the first step, returning 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. Figure 22-1 illustrates this step.

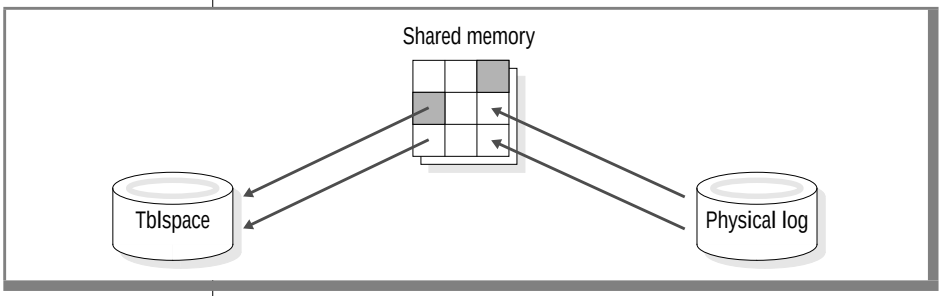


Figure 22-1
Writing All Remaining Before-Images in the Physical Log Back to Disk

Finding the Checkpoint Record in the Logical Log

In the second step, 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 address information needed to locate the most recent checkpoint record in the logical log is contained in the active PAGE_CKPT page of the root dbspace reserved pages. For more information, see "PAGE_CKPT" on page 38-9.

The address information from PAGE_CKPT also identifies the location of all logical-log records written after the most recent checkpoint. Figure 22-2 illustrates this step.

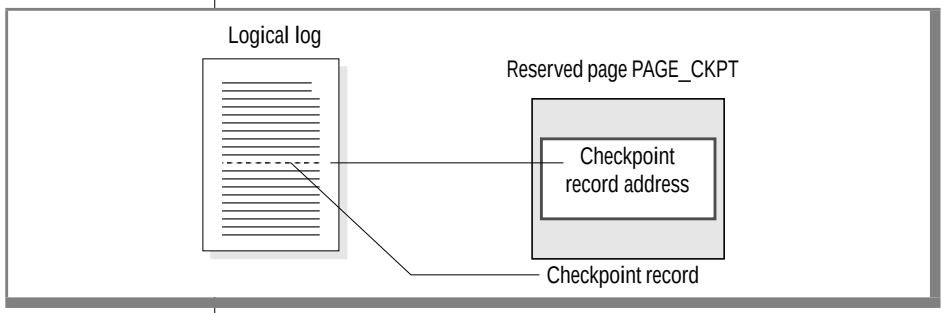


Figure 22-2
Locating the Most Recent Checkpoint Record in the Logical Log

Rolling Forward Logical-Log Records

The third step in fast recovery 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 where the uncontrolled shutdown occurred. Figure 22-3 illustrates this step.

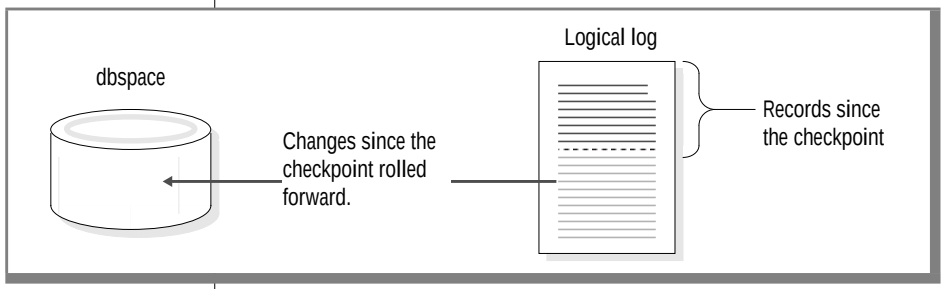


Figure 22-3
Rolling Forward the Logical-Log Records Written Since the Most Recent Checkpoint

Rolling Back Incomplete Transactions

The final step in fast recovery rolls back all logical-log records that are associated with transactions that were not committed at the time the system failed. (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 30-10.) This rollback procedure ensures that all databases are left in a consistent state.

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 contained within it are closed. Figure 22-4 illustrates the rollback procedure. When fast recovery is complete, the database server goes to quiescent or on-line mode

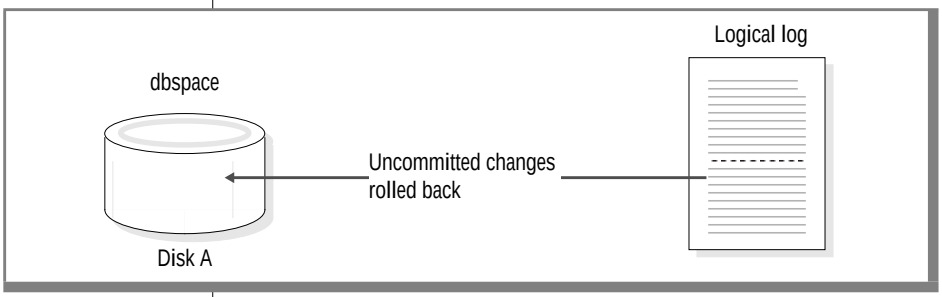


Figure 22-4
Rolling Back All Incomplete Transactions

Fault Tolerance

Section VI

Chapter

23

What Is Mirroring?

What Is Mirroring? 23-4
- What Are the Benefits of Mirroring? 23-4
- What Are the Costs of Mirroring? 23-5
- What Happens If You Do Not Mirror? 23-5
- What Should You Mirror? 23-5
- What Mirroring Alternatives Exist? 23-6

The Mirroring Process 23-7
- What Happens When You Create a Mirrored Chunk? 23-7
- What Are Mirror Status Flags? 23-8
- What Is Recovery? 23-8
- What Happens During Processing? 23-9
  - Disk Writes to Mirrored Chunks 23-9
  - Disk Reads from Mirrored Chunks 23-9
  - Detecting Media Failures 23-10
  - Recovering a Chunk 23-10
- What Happens If You Stop Mirroring? 23-11
- What Is the Structure of a Mirrored Chunk? 23-11

The first part of this chapter answers the following basic questions about the database server mirroring feature:

- What are the benefits of mirroring?
- What are the costs of mirroring?
- What happens if you do not mirror?
- What should you mirror?
- What mirroring alternatives exist?

The second part of the chapter discusses the actual mirroring process. The following aspects of the process are discussed:

- What happens when you create a mirrored chunk?
- What are the mirror status flags?
- What is recovery?
- What happens during processing?
- What happens if you stop mirroring?
- What is the structure of a mirrored chunk?

For instructions on how to perform mirroring tasks, refer to Chapter 24, "Using Mirroring."

For instructions on using the INFORMIX-Enterprise Command Center to perform mirroring tasks, refer to the *INFORMIX-Enterprise Command Center User Guide*.

What Is Mirroring?

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace or blobspace with an equal-sized *mirrored chunk*. Every write to the primary chunk is automatically accompanied by an identical write to the mirrored chunk. This concept is illustrated in Figure 23-1. If a failure occurs on the primary chunk, mirroring enables you to read from and write to the mirrored chunk until you can recover the primary chunk, all without interrupting user access to data.

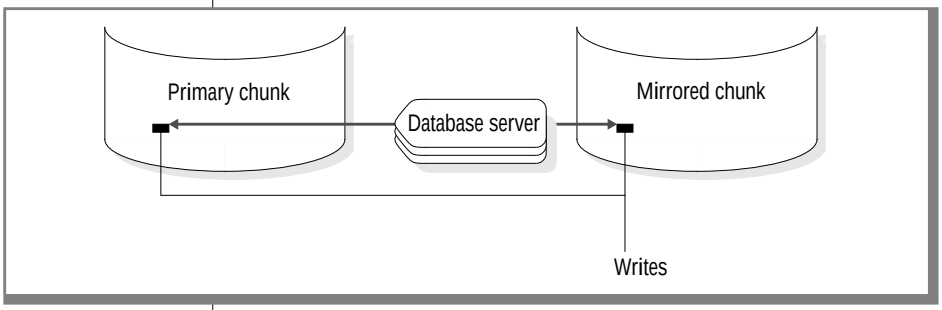


Figure 23-1
Writing Data to Both the Primary Chunk and the Mirrored Chunk

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.

What Are the Benefits of Mirroring?

In the event of a media failure, mirroring provides the database server administrator with a means of recovering data without having to take the database server off-line. 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 database that has extreme requirements for reliability in the face of media failure should be located in a mirrored dbspace. Above all, the root dbspace, which contains the database server reserved pages, should be mirrored.

What Are the 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 mirrored 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 mirrored 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 mirrored chunks, see "What Happens During Processing?" on page 23-9.

What Happens If You Do Not Mirror?

If you do not mirror your dbspaces, the frequency with which you have to restore from a dbspace backup in the event of a media failure increases.

When a mirrored chunk suffers a media failure, the database server reads exclusively from the chunk that is still on-line until you bring the down chunk back on-line. On the other hand, when an *unmirrored* 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 off-line 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. Unmirrored chunks that go down must be restored by recovering the dbspace from a backup.

What Should You 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 off-line 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 other chunks that store frequently used data. This action ensures that the activities of many users are not halted if one widely used chunk goes down.

What Mirroring Alternatives Exist?

Mirroring, as discussed in this manual, is a database server feature. Alternative mirroring solutions might be provided by your operating system or hardware.

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 mirrored chunks in parallel (see "Disk Writes to Mirrored Chunks" on page 23-9) helps to reduce the time required for this step. In addition, database server mirroring uses split reads to improve read performance. (See "Disk Reads from Mirrored Chunks" on page 23-9.) 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 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.

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 in a manner resilient to media failure. The disadvantage is that it is slower than database server mirroring for write operations.

The Mirroring Process

This section describes the mirroring process in greater detail. For instructions on how to perform mirroring operations such as creating mirrored chunks, starting mirroring, changing the status of mirrored chunks, and so on, refer to Chapter 24, "Using Mirroring."

What Happens When You Create a Mirrored Chunk?

When you specify a mirrored chunk, the database server copies all the data from the primary chunk to the mirrored 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 mirrored chunk, from the root dbspace reserved pages to the dbspace backup. If you perform a data restore, the updated configuration information at the beginning of the dbspace 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 dbspace 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 once the backup has begun.

For more information on creating mirrored chunks, refer to Chapter 24, "Using Mirroring."

What Are Mirror Status Flags?

Dbspaces and blobspaces have status flags that indicate whether the dbspace or blobspace is mirrored, unmirrored, or mirrored but requiring 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 mirrored chunk
- Whether the chunk is currently on-line, down, a new mirrored chunk that requires a level-0 backup of the root dbspace, or being recovered

For descriptions of these chunk status flags, refer to "-d Option" on page 35-72. For information on how to display these status flags, refer to "Monitoring Disk Usage" on page 29-53.

What Is Recovery?

When the database server recovers a mirrored 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 on-line chunk onto the new, repaired chunk until the two are considered identical.

When you initiate recovery, the database server puts the down chunk in recovery mode and copies the information from the on-line chunk to the recovery chunk. When the recovery is complete, the chunk automatically receives on-line status. You perform the same steps whether you are recovering the primary chunk of a mirrored pair or recovering the mirrored chunk.

Tip: *You can still use the on-line chunk while the recovery process is occurring. 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 "Recovering a Mirrored Chunk" on page 24-11.

What Happens During Processing?

This section discusses some of the details of disk I/O for mirrored chunks and how the database server handles media failure for these chunks.

Disk Writes to Mirrored Chunks

During database server processing, the database server performs mirroring by executing two writes for each modification: one to the primary chunk and one to the mirrored chunk. Virtual processors of the AIO class perform the actual disk I/O. For more information, refer to "Asynchronous I/O" on page 9-26.

The requesting thread submits the two write requests (one for the primary chunk and one for the mirrored chunk) asynchronously. That is, if two AIO virtual processors are idle, they can perform the two disk writes in parallel. In the meantime, the requesting thread can perform any additional processing that does not depend on the result of the mirror I/O.

Disk Reads from Mirrored 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 mirrored 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 23-2 on page 23-10 illustrates a split read.

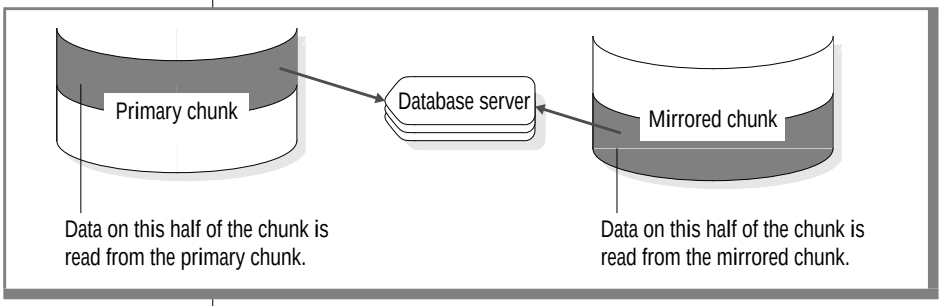


Figure 23-2
Split Read Reducing the Maximum Distance Over Which the Disk Head Must Travel

Detecting 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 "What Are Mirror Status Flags?" on page 23-8.

If the database server detects that a primary (or mirror) chunk device has failed, reads and writes continue for the one chunk that remains on-line. This statement is true even if the administrator intentionally brings down one of the chunks.

Once the administrator recovers the down chunk and returns it to on-line status, reads are again split between the primary and mirrored chunks, and writes are made to both chunks.

Recovering a Chunk

The database server uses asynchronous I/O to minimize the time required for recovering a chunk. The read from the chunk that is on-line 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.

What Happens If You Stop Mirroring?

When you end mirroring, the database server immediately frees the mirrored 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.

What Is the Structure of a Mirrored Chunk?

The mirrored chunk contains the same control structures as the primary chunk. Mirrors of blobspace chunks contain blobspace overhead pages; mirrors of dbspace chunks contain dbspace overhead pages. For information on these structures, refer to "Structure of a Mirrored Chunk" on page 38-17.

A display of disk-space use, provided by one of the methods discussed under "Monitoring Chunks" on page 29-53, always indicates that the mirrored chunk is full, even if the primary chunk has free space. The *full* mirrored 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 mirrored chunk are on-line.

If the primary chunk goes down, and the mirrored chunk becomes the primary chunk, disk-space allocation reports then accurately describe the fullness of the new primary chunk.

Chapter

24

Using Mirroring

Steps Required for Mirroring Data 24-3

Enabling Mirroring 24-4
Enabling Mirroring with ON-Monitor 24-5
Enabling Mirroring by Editing the ONCONFIG File 24-5

Allocating Disk Space for Mirrored Data 24-5

Starting Mirroring 24-6
Mirroring the Root Dbspace During Initialization 24-7
Setting MIRRORPATH and MIRROROFFSET with ON-Monitor 24-7
Setting MIRRORPATH and MIRROROFFSET with a Text Editor 24-7
Starting Mirroring for Unmirrored Dbspaces 24-8
Starting Mirroring for Unmirrored Dbspaces with ON-Monitor 24-8
Starting Mirroring for Unmirrored Dbspaces with onspaces 24-8
Starting Mirroring for New Dbspaces 24-8
Starting Mirroring for New Dbspaces with ON-Monitor . . . 24-9
Starting Mirroring for New Dbspaces with onspaces 24-9

Adding Mirrored Chunks 24-9
Adding Mirrored Chunks with ON-Monitor 24-9
Adding Mirrored Chunks with the onspaces Utility 24-9

Changing the Mirror Status 24-10
Taking Down a Mirrored Chunk 24-10
Taking Down a Mirrored Chunk with ON-Monitor 24-10
Taking Down a Mirrored Chunk with the onspaces Utility. 24-11
Recovering a Mirrored Chunk. 24-11
Recovering a Mirrored Chunk with ON-Monitor 24-11
Recovering a Mirrored Chunk with onspaces 24-11

Relinking a Chunk to a Device After a Disk Failure 24-12

Ending Mirroring 24-12
Ending Mirroring with ON-Monitor 24-13
Ending Mirroring with onspaces 24-13

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. Then it describes the following tasks:

- Enabling mirroring
- Allocating disk space for mirrored chunks
- Starting mirroring (creating mirrored chunks)
- Adding chunks to mirrored dbspaces
- Changing the mirror status of chunks
- Relinking mirrored chunks after a disk failure
- Ending mirroring

Steps Required for Mirroring Data

To start mirroring data on a database server that is not running with the mirroring function enabled, you must perform the following steps:

1. Take the database server off-line and enable mirroring. See "Enabling Mirroring" on page 24-4.
2. Reinitialize shared memory.
3. Allocate disk space for the mirrored chunks. You can allocate this disk space at any time, as long as the disk space is available when you specify mirrored chunks in the next step. See "Allocating Disk Space for Mirrored Data" on page 24-5.
4. Choose the dbspace that you want to mirror, and create mirrored chunks by specifying a mirror-chunk pathname and offset for each primary chunk in that dbspace. The mirroring process starts after you perform this step. Repeat this step for all the dbspaces that you want to mirror. See "Starting Mirroring" on page 24-6.

Enabling Mirroring

When you enable mirroring, you invoke 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 mirrored chunks for a dbspace or blobspace. See "Starting Mirroring" on page 24-6.

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.

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 dbspace, you can change the value of the MIRROR parameter through ON-Monitor or by editing your configuration file.

You can change the value of MIRROR while the database server is in on-line mode, but the change does not take effect until you reinitialize shared memory (take the database server off-line and then to quiescent or on-line mode).

If you are logged in on a UNIX system as user **informix** or **root**, or on a Windows NT system as a member of the **Informix-Admin** group, you can change the value of MIRROR either by using ON-Monitor or by editing the ONCONFIG file with a text editor. Informix recommends that you enable mirroring by editing the ONCONFIG file. If you are not careful when you enable mirroring with ON-Monitor, you might accidentally reinitialize your disk.

Enabling Mirroring with ON-Monitor

To enable mirroring, choose **Parameters→Initialize**. In the field that is labelled **Mirror**, enter Y. Press ESC to record changes.

A series of screens appears displaying other system parameters. Type ESC at each screen to maintain the same values. 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.*

Reinitialize shared memory (take the database server off-line and then to quiescent mode) for the change to take effect.

Enabling Mirroring by Editing the ONCONFIG File

Edit the ONCONFIG file. Change the value of MIRROR to 1. Reinitialize shared memory (take the database server off-line and then to quiescent mode) for the change to take effect.

Allocating Disk Space for Mirrored Data

Before you can create a mirrored chunk, you must allocate disk space for this purpose. You can allocate either raw disk space or cooked file space for mirrored chunks. For a discussion of allocating disk space, refer to "Allocating Disk Space" on page 14-4.

Always allocate disk space for a mirrored chunk on a different disk than the corresponding primary chunk with, ideally, a different controller. This setup allows you to access the mirrored chunk if the disk on which the primary chunk is located goes down, or vice versa.

Use the UNIX link (**ln**) command to link the actual files or raw devices of the mirrored chunks to mirror pathnames. In the event of disk failure, you can link a new file or raw device to the pathname, eliminating the need to physically replace the disk that failed before the chunk is brought back on-line. (See "Relinking a Chunk to a Device After a Disk Failure" on page 24-12.)

Starting Mirroring

Mirroring starts when you create a mirrored chunk for each primary chunk in a dbspace or blobspace. This action consists of specifying disk space that you have already allocated, either raw disk space or a cooked file, for each mirrored chunk. You can use either ON-Monitor or the **onspaces** utility to create mirrored chunks.

When you create a mirrored chunk, the database server performs the *recovery* process, copying data from the primary chunk to the mirrored 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 perform the recovery process immediately after you create the mirrored chunk but waits until you perform a level-0 dbspace backup. For an explanation of this behavior see "What Happens When You Create a Mirrored Chunk?" on page 23-7.

You must always start mirroring for an entire dbspace or blobspace. The database server does not permit you to select particular chunks in a dbspace or blobspace to mirror. When you select a space to mirror, you must create mirrored chunks for every chunk within the space.

You start mirroring a dbspace 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 or blobspace

Each of these operations requires you to create mirrored chunks for the existing chunks in the dbspace or blobspace. You can perform all three operations with ON-Monitor, and you can perform the last two with **onspaces** as well.

Mirroring the Root Dbspace During Initialization

If you enable mirroring when you initialize the database server, you can also specify a mirror pathname and offset for the root chunk. The database server creates the mirrored chunk when it is initialized. However, because the root chunk contains logical-log files, mirroring does not actually start until you perform a level-0 dbspace backup. (See "What Happens When You Create a Mirrored Chunk?" on page 23-7.)

To specify the root mirror pathname and offset, set the configuration parameters MIRRORPATH and MIRROROFFSET.

If you do not provide a mirror pathname and offset, but you do want to start mirroring the root dbspace, you must change the mirroring status of the root dbspace once the database server is initialized. See "Starting Mirroring for Unmirrored Dbspaces" on page 24-8.

Setting MIRRORPATH and MIRROROFFSET with ON-Monitor

If you are using ON-Monitor to initialize the database server, you can set the MIRRORPATH and MIRROROFFSET parameters in the DISK PARAMETERS screen of the Parameters menu, Initialize option.

Setting MIRRORPATH and MIRROROFFSET with a Text Editor

If you are using **oninit** to initialize the database server, you must use a text editor to set the values of MIRRORPATH and MIRROROFFSET in the ONCONFIG file before you bring up the database server.

Starting Mirroring for Unmirrored Dbspaces

You can start mirroring for any dbspace or blobspace with either ON-Monitor or the **onspaces** utility.

Starting Mirroring for Unmirrored Dbspaces with ON-Monitor

Use the Mirror option of the Dbspaces menu to start mirroring a dbspace. The first screen displays a list of dbspaces. To select the dbspace that you want to mirror, move the cursor down the list to the correct dbspace and type CTRL-B. The Mirror option then displays a screen for each chunk in the dbspace. You can enter a mirror pathname and offset in this screen. After you enter information for each chunk, press ESC to exit the option. The database server recovers the new mirrored chunks unless they contain logical-log files, in which case recovery is postponed until after you create a level-0 dbspace backup.

Starting Mirroring for Unmirrored Dbspaces with onspaces

You can also use the **onspaces** utility to start mirroring a dbspace or blobspace. 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
```

For a full description of the **onspaces** syntax, see “onspaces: Manage Database Spaces” on page 35-48.

Starting Mirroring for New Dbspaces

You can also start mirroring when you create a new dbspace or blobspace. You can use either ON-Monitor or the **onspaces** utility to do this.

Starting Mirroring for New Dbspaces with ON-Monitor

To create a dbspace with mirroring, choose the Create option of the Dbspaces menu. This option displays a screen in which you can specify the pathname, offset, and size of a primary chunk and the pathname and offset of a mirrored chunk for the new dbspace.

Starting Mirroring for New Dbspaces with 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 mirrored chunk **/dev/mirror_chk1**:

```
% onspaces -c -d db_acct -p /dev/chunk1 -o 0 -s 2500 -m /dev/mirror_chk1 0
```

For a full description of the **onspaces** syntax, see "onspaces: Manage Database Spaces" on page 35-48.

Adding Mirrored Chunks

If you add a chunk to a dbspace that is mirrored, you must also add a corresponding mirrored chunk.

Adding Mirrored Chunks with ON-Monitor

In ON-Monitor, the Add-chunk option of the Dbspaces menu displays fields in which to enter the primary-chunk pathname, offset, and size, and the mirror-chunk pathname and offset.

Adding Mirrored Chunks with the onspaces Utility

You can also use the **onspaces** utility to add a primary chunk and its mirrored chunk to a dbspace. The following example adds a chunk, **chunk2**, to the **db_acct** dbspace. Because the dbspace is mirrored, a mirrored chunk, **mirror_chk2**, is also added.

```
% onspaces -a db_acct -p /dev/chunk2 -o 5000 -s 2500 -m /dev/mirror_chk2 5000
```

For a full description of the **onspaces** syntax, see "onspaces: Manage Database Spaces" on page 35-48.

Changing the Mirror Status

You can make the following two changes to the status of a mirrored chunk:

- Change a mirrored chunk from on-line to down
- Change a mirrored 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 mirrored chunk, as long as the other chunk in the pair is on-line.

For information on how to determine the status of a chunk, refer to "Monitoring Disk Usage" on page 29-53.

Taking Down a Mirrored Chunk

When a mirrored chunk is *down*, the database server cannot write to it or read from it. You might take down a mirrored chunk to relink the chunk to a different device. (See "Relinking a Chunk to a Device After a Disk Failure" on page 24-12.) 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 mirrored chunks for that dbspace.

Taking Down a Mirrored Chunk with ON-Monitor

To use ON-Monitor to take down a mirrored chunk, choose the Status option from the Dbspaces menu. 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).

Taking Down a Mirrored Chunk with the onspaces Utility

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
```

For a full description of the **onspaces** syntax, see "onspaces: Manage Database Spaces" on page 35-48.

Recovering a Mirrored Chunk

You recover a down chunk to begin mirroring the data in the chunk that is on-line.

Recovering a Mirrored Chunk with ON-Monitor

To use ON-Monitor to recover a down chunk, choose the Status option from the Dbspaces menu. With the cursor on the dbspace that contains the down chunk, press F3 or CTRL-B. The system displays a screen that lists all the chunks in the dbspace. Move the cursor to the chunk that is down, and type F3 or CTRL-B to recover it.

Recovering a Mirrored Chunk with onspaces

You can also use the **onspaces** utility to recover a down chunk. For example, to recover a chunk that has the pathname **/dev/mirror_chk1** and an offset of 0 kilobytes, issue the following command:

```
% onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -O
```

For a full description of the **onspaces** syntax, see "onspaces: Manage Database Spaces" on page 35-48.

UNIX

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 mirrored chunk before the disk that failed is brought back on-line. 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 -lg
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 mirrored chunk pathname 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
```

Ending Mirroring

When you end mirroring for a dbspace, the database server immediately releases the mirrored chunks of that dbspace. These chunks are immediately available for reassignment to other dbspaces or blobspaces. Only users **informix** and **root** on a UNIX system or members of the **Informix-Admin** group on a Windows NT system can initiate this action.

You cannot end mirroring if any of the primary chunks in the dbspace are down. The system can be in on-line mode when you end mirroring.

Ending Mirroring with ON-Monitor

To end mirroring for a dbspace or blobspace with ON-Monitor, select the Mirror option of the Dbspaces menu. Select a dbspace or blobspace that is mirrored, and type CTRL-B or F3.

Ending Mirroring with onspaces

You can also end mirroring with the **onspaces** utility. For example, to end mirroring for the root dbspace, enter the following command:

```
% onspaces -r rootdbs
```

For a full description of the **onspaces** syntax, see "onspaces: Manage Database Spaces" on page 35-48.

Chapter

25

What Is High-Availability Data Replication?

What Is Data Replication? 25-4
What Is High-Availability Data Replication? 25-4
What Are Primary and Secondary Database Servers? 25-5
How Is High-Availability Data Replication Different from Mirroring? 25-6
How Is High-Availability Data Replication Different from Two-Phase Commit? 25-7

How Does High-Availability Data Replication Work? 25-8
How Does Data Initially Replicate? 25-8
Reproducing Updates to the Primary Database Server 25-9
How Are the Log Records Sent? 25-9
What Are the High-Availability Data-Replication Buffers? . . 25-10
When Are Log Records Sent? 25-10
Synchronous Updating 25-11
Asynchronous Updating 25-11
What Threads Handle High-Availability Data Replication? . . . 25-13
Checkpoints Between Database Servers 25-14
How Is Data Synchronization Tracked? 25-14

HIgh-Availability Data-Replication Failures 25-15
What Are High-Availability Data-Replication Failures? 25-15
How Are High-Availability Data-Replication Failures Detected? 25-16
What Happens When a High-Availability Data-Replication Failure Is Detected? 25-16
Considerations After High-Availability Data-Replication Failure 25-17
Actions to Take If the Secondary Database Server Fails 25-18
Actions to Take If the Primary Database Server Fails 25-18

Redirection and Connectivity for Data-Replication Clients 25-22
Designing Clients for Redirection 25-23
Automatic Redirection with DBPATH 25-23
How Does the DBPATH Redirection Method Work? 25-23
What Does the Administrator Need to Do? 25-24
What Does the User Need to Do? 25-24
Administrator-Controlled Redirection with the Connectivity Information 25-25
How Does the Connectivity Information-Redirection Method Work? 25-25
What Does the Administrator Need to Do? 25-26
What Does the User Need to Do? 25-29
User-Controlled Redirection with INFORMIXSERVER 25-29
How Does the INFORMIXSERVER Redirection Method Work? 25-29
What Does the Administrator Need to Do? 25-30
What Does the User Need to Do? 25-30
Handling Redirection Within an Application. 25-30
A Connection Loop and Database Server Type Check 25-31
Comparison of Different Redirection Mechanisms 25-33

Designing High-Availability Data-Replication Clients 25-34
Setting Lock Mode to Wait for Access to Primary Database Server 25-34
Designing Clients to Use the Secondary Database Server 25-35
No Data Modification Statements 25-35
Locking and Isolation Level 25-37
Using Temporary Dbspaces for Sorting and Temporary Tables 25-38

This chapter describes Informix high-availability data replication.

High-availability data replication provides synchronous and asynchronous data replication of an entire database server. Use high-availability data replication if you require a hot standby. If you want to use asynchronous data replication, see the *Guide to Informix Enterprise Replication*.

The chapter discusses the following topics:

- What data replication is, both in a broad sense and in the context of the database server
- How high-availability data replication works
- How high-availability data replication handles failures
- How the system administrator or user can redirect a client to connect to the other database server in the high-availability data-replication pair
- What the design considerations are for applications that connect to the secondary database server

A companion chapter, Chapter 26, "Using High-Availability Data Replication," contains instructions on how to accomplish the administrative tasks that are involved in using high-availability data replication.

What Is Data Replication?

Data replication, in the broadest sense, refers to the process of representing database objects at more than one distinct site.

For example, 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.

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, error prone, and difficult to maintain. Instead, the concept of data replication is often coupled with *replication transparency*. Replication transparency is functionality built into a database server (instead of into client applications) to handle automatically the details of locating and maintaining data replicas.

What Is High-Availability Data Replication?

Within the broad framework of data replication, the database server implements nearly transparent data replication of entire database servers. All the data managed by one database server is replicated and dynamically updated on another database server, often at a separate geographical location. High-availability data replication 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.

What Are Primary and Secondary Database Servers?

When you configure a pair of database servers to use high-availability data replication, one database server is called the *primary* database server, and the other is called the *secondary* database server. (In the context of high-availability data replication, a database server that does not use high-availability data replication is referred to as a *standard* database server.)

During normal operation, clients can connect to the primary database server and use it as they would an ordinary database server. Clients can also use the secondary database server during normal operation, but only to read data. The secondary database server does not permit updates from client applications.

As illustrated in Figure 25-1, the secondary database server is dynamically updated, with changes made to the data managed by the primary database server.

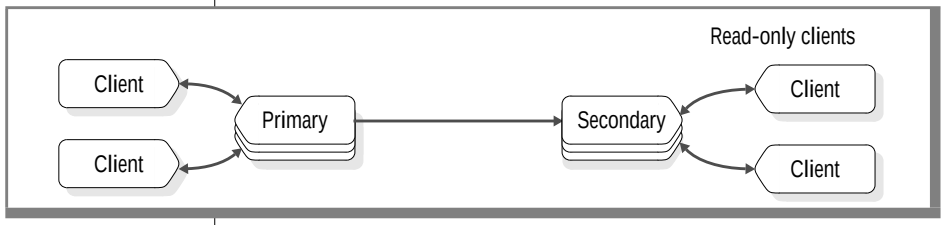


Figure 25-1
A Primary and Secondary Database Server in a High-Availability Data-Replication Pair

If one of the database servers fails, as shown in Figure 25-2, you can redirect the clients that use that database server to the other database server in the pair.

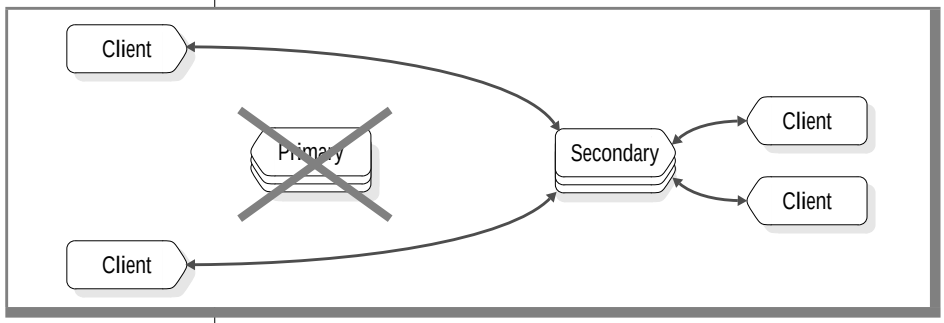


Figure 25-2
Database Servers in a High-Availability Data-Replication Pair and Clients After a Failure

If a primary database server fails, you can change the secondary database server to a standard database server so that it can accept updates.

High-availability data replication has the following features:

- Provides for quick recovery if one database server experiences a failure
- Allows for load balancing across the two database servers

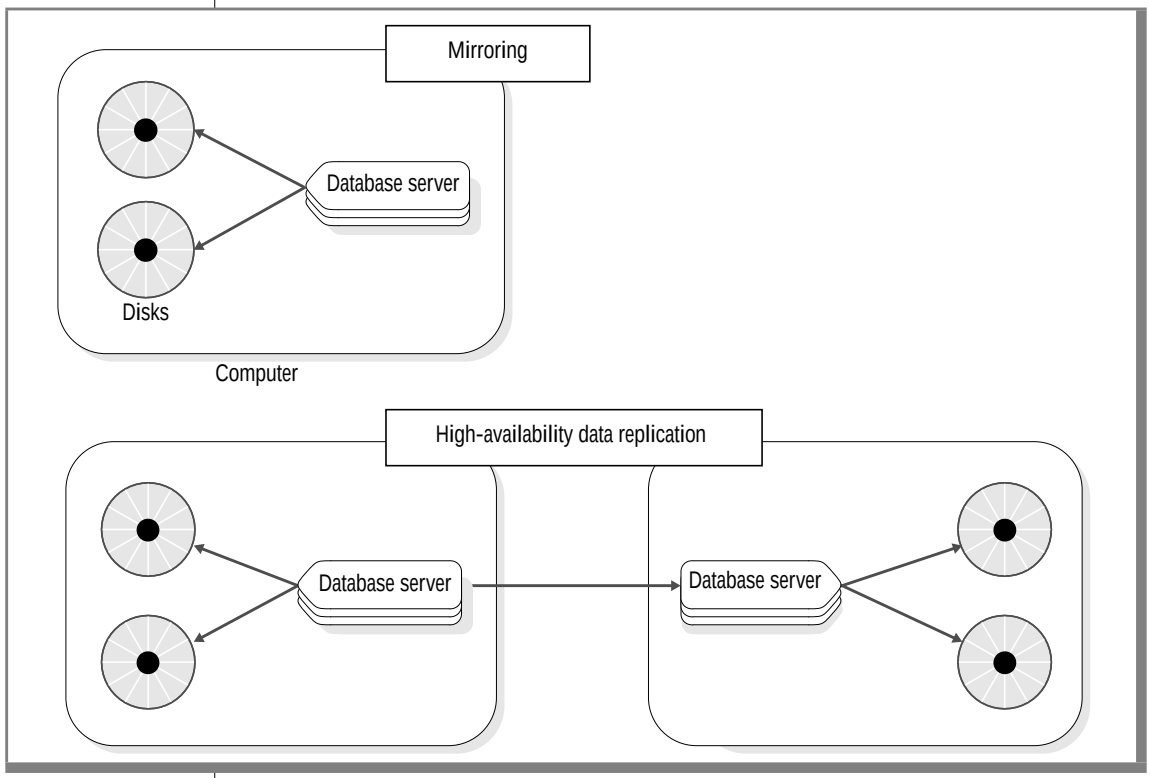
How Is High-Availability Data Replication Different from Mirroring?

High-availability data replication and mirroring are both transparent ways of making the database server more fault tolerant. However, as shown in Figure 25-3 on page 25-7, they are quite different.

Mirroring, described in "What Is Mirroring?" on page 23-4, 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.

High-availability data replication, on the other hand, duplicates all the data managed by a database server (not just specified dbspaces) on an entirely separate database server. Because high-availability data replication involves two separate database servers, the data that these database servers manage is protected against all types of database server failures, such as a computer crash or the catastrophic failure of an entire site, not just disk failures.

Figure 25-3
A Comparison of Mirroring and High-Availability Data Replication



How Is High-Availability Data Replication Different from Two-Phase Commit?

The two-phase commit protocol, described in detail in Chapter 30, "Multiphase Commit Protocols," 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 dynamic high-availability data replication.

How Does High-Availability Data Replication Work?

This section describes the mechanisms that the database server uses to perform high-availability data replication. For instructions on how to set up, start, and administer a high-availability data-replication system, refer to Chapter 26, "Using High-Availability Data Replication."

How Does Data Initially Replicate?

The database server uses dbspace backups and logical-log files (both those backed up to tape and those on disk) to do an initial replication of the data on one database server to a second database server. The procedure is basically as follows:

1. To make the bulk of the data managed by the two database servers the same, create a level-0 backup of all the dbspaces on one database server, and restore all the dbspaces from that dbspace backup on the other database server in the data-replication pair.
2. The database server that you restored from a dbspace backup in the first step then reads all the logical-log records generated since that dbspace backup from the database server on which the dbspace backup was created. 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 on performing the preceding steps, refer to "Starting High-Availability Data Replication for the First Time" on page 26-11.

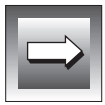
You must perform the initial high-availability data replication with a dbspace backup. You cannot use data-migration utilities such as **onload** and **onunload** to replicate data because the physical page layout of tables on each database server must be identical in order for high-availability data replication to work.

In the preceding steps, the database server from which you create the dbspace backup can be the primary database server or the secondary database server.

When high-availability data replication is working, the primary database server is in on-line mode and accepts updates and queries just as a standard database server does. The secondary database server is in logical-recovery mode and cannot accept SQL statements that result in writes to disk (except for sorting and temporary tables).

Reproducing Updates to the Primary Database Server

High-availability data replication reproduces updates to the primary database server on the secondary database server by having the primary database server send all its logical-log records to the secondary database server as they are generated. (For general information on transaction logging, refer to "What Is Transaction Logging?" on page 16-8.) The secondary database server receives the logical-log records generated on the primary database server and applies them to its dbspaces.



Important: *The database server cannot replicate updates to databases that do not use transaction logging. The database server does not replicate data in blobspaces either.*

How Are the Log Records Sent?

As shown in Figure 25-4 on page 25-10, 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 secondary database server.

The 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 using logical recovery.

What Are the High-Availability Data-Replication Buffers?

The high-availability data-replication buffers are part of the virtual shared memory managed by the primary database server. The high-availability data-replication buffers hold logical-log records before the primary database server sends them to the secondary database server. The high-availability data-replication buffers are the same size as the logical-log buffers. Figure 25-4 shows this concept.

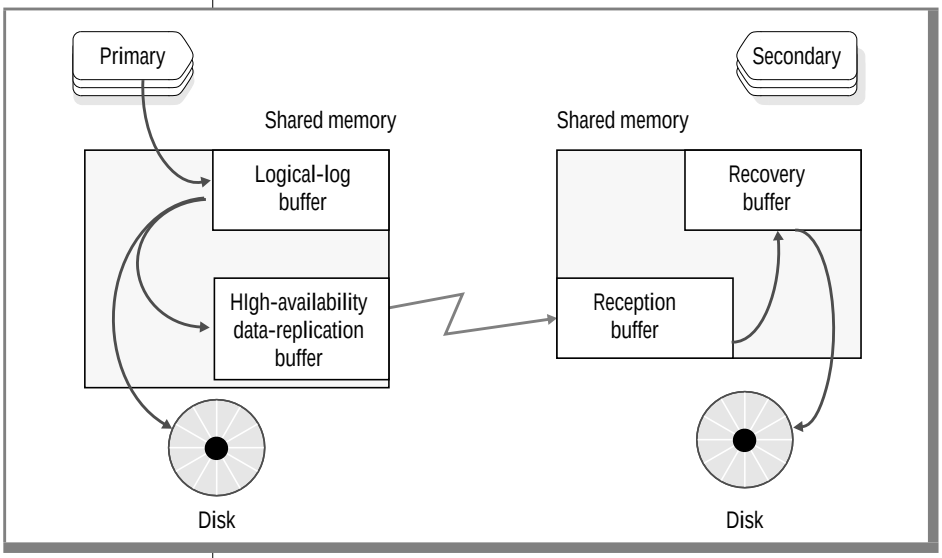


Figure 25-4
How Logical-Log Records Are Sent from the Primary Database Server to the Secondary Database Server

When Are Log Records Sent?

The primary database server sends the contents of the high-availability data-replication buffer to the secondary database server either *synchronously* or *asynchronously*. The value of the ONCONFIG configuration parameter DRINTERVAL, described on page 33-23, determines whether the database server uses synchronous or asynchronous updating.

Synchronous Updating

If you set DRINTERVAL to -1, high-availability data replication occurs *synchronously*. As soon as the primary database server writes the logical-log buffer contents to the high-availability data-replication buffer, it sends those records from the high-availability data-replication buffer to the secondary database server. The logical-log buffer flush on the primary database server completes only after the primary database server receives acknowledgment from the 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 secondary database server if a failure occurs.

Asynchronous Updating

If you set DRINTERVAL to anything other than -1, data replication occurs *asynchronously*. The primary database server flushes the logical-log buffer after it copies the logical-log buffer contents to the high-availability data-replication buffer. Independent of that action, the primary database server sends the contents of the high-availability data-replication buffer across the network when one of the following conditions occurs:

- The high-availability data-replication buffer becomes full.
- An application commits a transaction on an unbuffered database.
- The time interval, specified by the ONCONFIG parameter DRINTERVAL on the primary database server, has elapsed since the last time records were sent to the secondary database server.

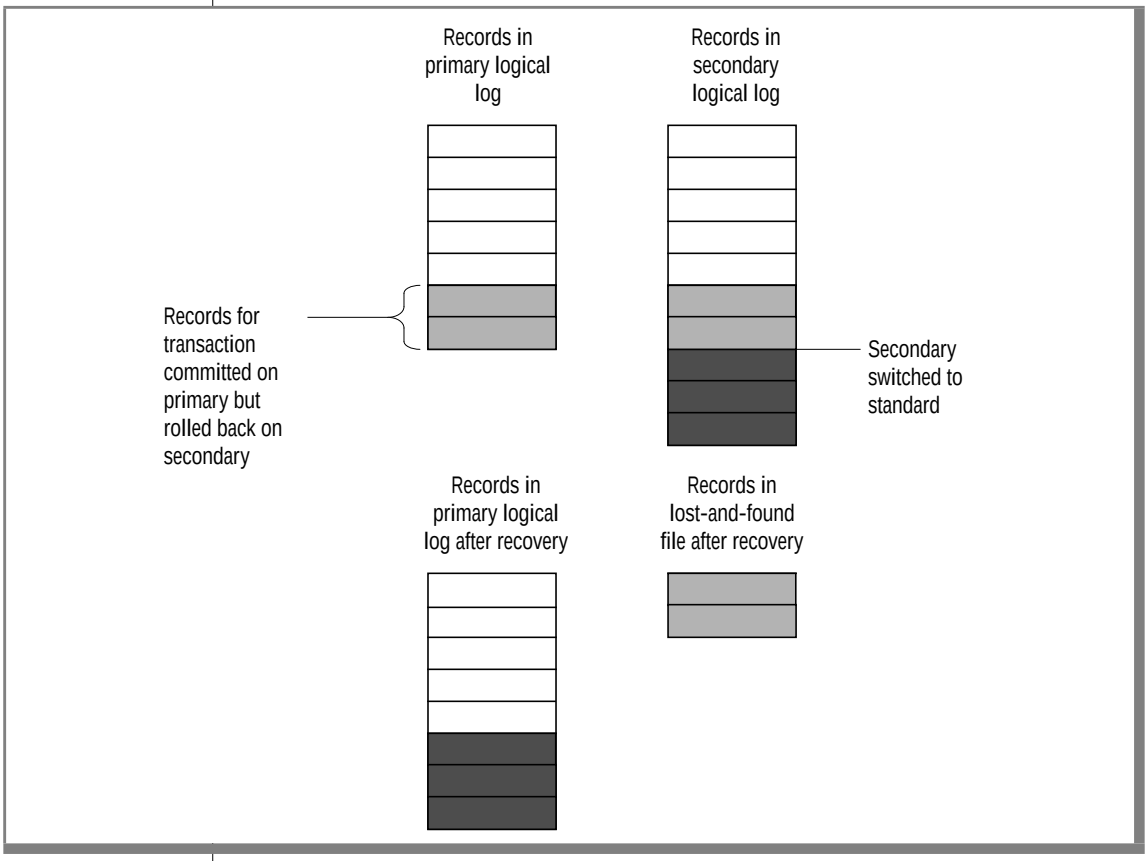
This method of updating might provide better performance than synchronous updating. However, as explained in the following section, transactions might be lost.

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 occur if a failure happens after the primary database server copies a commit record to the high-availability data-replication 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 (specified by the ONCONFIG parameter DRLOSTFOUND) during logical recovery of the primary database server. Figure 25-5 illustrates the process.

Figure 25-5
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 primary database server that is writing the transaction records to disk and the primary database server that is sending these records to the secondary database server.

What Threads Handle High-Availability Data Replication?

The database server starts specialized threads to support data replication. As shown in Figure 25-6, a thread called **drprsend** on the primary database server sends the contents of the high-availability data-replication buffer across the network to a thread called **drsecrcv** on the secondary database server.

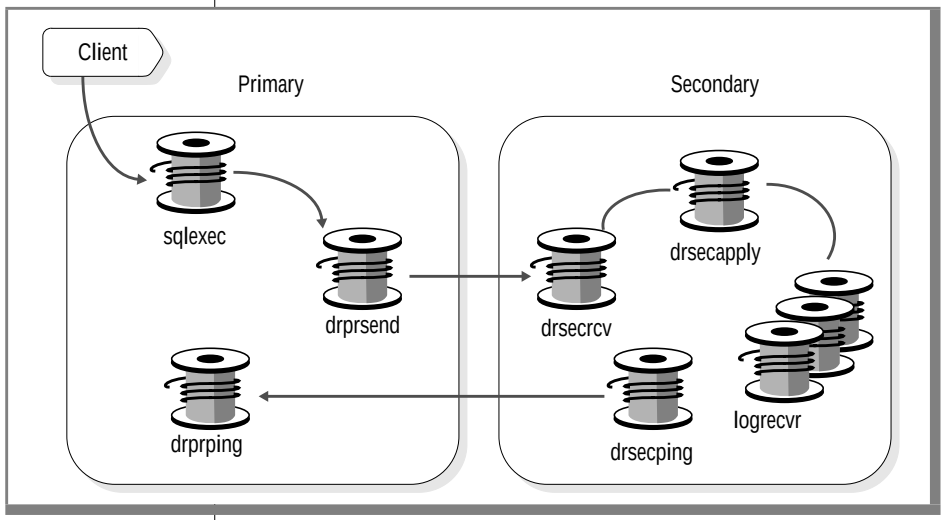


Figure 25-6 *Threads That Manage Data Replication*

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 managed by the secondary database server. The ONCONFIG parameter ON_RECVRY_THREADS specifies the number of **logrecvr** threads used.

The remaining threads that the database server starts for high-availability data replication are the **drprping** and **drsecping** threads, which are responsible for sending and receiving the signals that indicate if the two database servers are connected.

Checkpoints Between Database Servers

Checkpoints between database servers in a high-availability data-replication pair are synchronous, regardless of the value of DRINTERVAL. (See "Checkpoints" on page 11-57.) A checkpoint on the primary database server completes only after it completes on the secondary database server. If the checkpoint does not complete within the time specified by the ONCONFIG parameter DRTIMEOUT, the primary database server assumes that a failure has occurred. See "What Are High-Availability Data-Replication Failures?" on page 25-15.

How Is Data Synchronization Tracked?

To keep track of synchronization, each database server in the pair keeps track of the following information in its archive reserve page (described in "PAGE_ARCH" on page 38-14):

- 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.

HIgh-Availability Data-Replication Failures

This section discusses the causes and consequences of a high-availability data-replication failure, as well as the administrator's options for managing failure and restarting data replication.

What Are High-Availability Data-Replication Failures?

A high-availability data-replication failure is a loss of connection between the database servers in a high-availability data-replication pair. Any of the following situations could 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
- An administrative action to turn data replication off on one of the database servers (that is, changing the type of the database server to standard)
- A disk failure on the secondary database server that is not resolved by a mirrored chunk

Tip: *A high-availability data-replication failure does not necessarily mean that one of the database servers has failed, only that the high-availability data-replication connection between the two database servers is lost.*

How Are High-Availability Data-Replication Failures Detected?

The database server interprets either of the following conditions as a high-availability data-replication failure:

- A specified time-out value was exceeded.

  In the course of normal high-availability data-replication 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 specified by DRTIMEOUT, the database server assumes that a high-availability data-replication failure has occurred.

- The periodic signaling (*pinging*) of the other database server over the network does not yield response.

  Both database servers send a signal (or *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. The database servers signal each other regardless of whether the primary database server sends any records to the secondary database server. If a database server does not respond to four signal attempts in a row, the database server that was signaling assumes that a high-availability data-replication failure has occurred.

What Happens When a High-Availability Data-Replication Failure Is Detected?

After a database server detects a high-availability data-replication failure, it writes a message to its message log (for example, `DR: receive error`) and turns data replication off. Thus, the high-availability data-replication connection between the two database servers is dropped. Both database servers experience the high-availability data-replication connection being dropped.

If the secondary database server remains on-line after a high-availability data-replication failure, and the configuration parameter DRAUTO is set to 1 (RETAIN_TYPE) or 2 (REVERSE_TYPE), the type of that database server changes automatically to standard. (For more information, see "What Is Automatic Switchover?" on page 25-19). If DRAUTO is set to 0 (OFF), the secondary database server attempts to reestablish communication with the primary database server periodically.

Considerations After High-Availability Data-Replication Failure

Consider the following two issues when a high-availability data-replication failure occurs:

- How the clients should react to the failure

  If the failure is a real failure (and not due to 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. How to redirect clients is explained in "Redirection and Connectivity for Data-Replication Clients" on page 25-22.

- How the database servers should react to the failure

  Which administrative actions to take after a high-availability data-replication failure depends on whether the primary database server or the secondary database server failed. This topic is discussed in the following sections: "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 in on-line mode.

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 25-22. 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.

Restarting After the Secondary Database Server Fails

The steps in restarting data replication after a failure of the secondary database server are listed in "Restarting If the Secondary Database Server Fails" on page 26-29.

Actions to Take If the Primary Database Server Fails

If the primary database server fails, the secondary database server can behave in the following three ways:

- The secondary database server can remain in logical-recovery mode. In other words, no action is taken. This would be the case if you expect the high-availability data-replication 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 remain in logical-recovery mode, awaiting *manual switchover*.

Automatic switchover and manual switchover are described in the following sections.

What Is Automatic Switchover?

Automatic switchover means that the secondary database server automatically becomes a standard database server after it detects a high-availability data-replication failure. It first rolls back any open transactions and then comes into on-line mode as a standard 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 database server, you must be sure that it has enough logical-log disk space to allow processing to continue without backing up logical-log files *or* that 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. To redirect clients, use any of the mechanisms described in "Redirection and Connectivity for Data-Replication Clients" on page 25-22.

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 manually switching the secondary database server to a standard database server.

The main disadvantage of automatic switchover is that it requires a very stable network to function appropriately. This issue is discussed in "Using Automatic Switchover Without a Reliable Network" on page 25-21.

Restarting Data Replication After Automatic Switchover

The steps required to restart data replication after an automatic switchover are listed in "The Secondary Database Server Is Changed to a Standard Database Server Automatically" on page 26-33.

When you succeed in bringing the original primary database server back online, the high-availability data-replication 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-standard database server switches directly to type primary. 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. Both scenarios (DRAUTO set to RETAIN_TYPE and DRAUTO set to REVERSE_TYPE) are shown in Figure 25-7 on page 25-20.

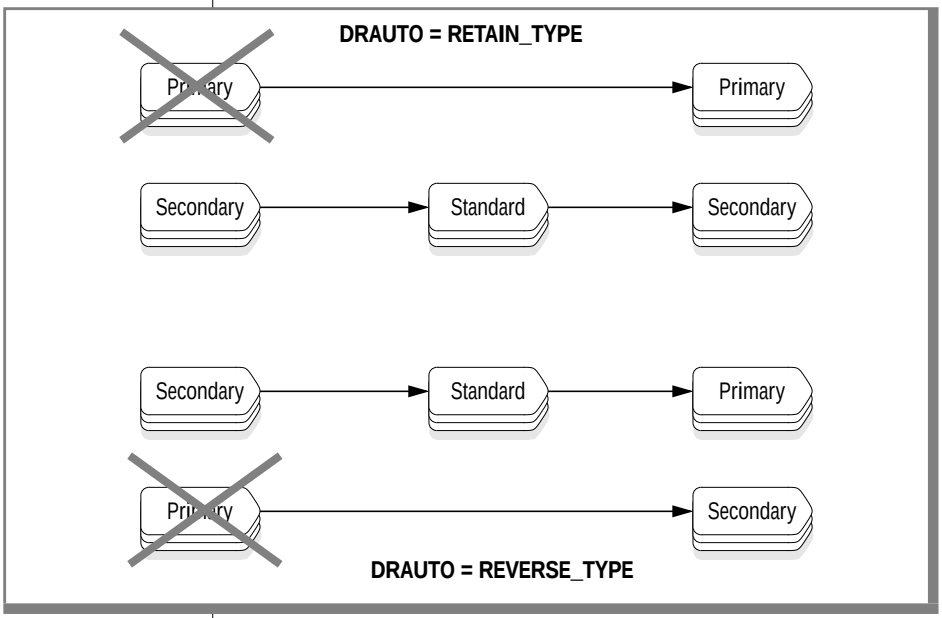


Figure 25-7
Automatic Switchover After a Failure on the Primary Database Server

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 would happen if the primary database server did not actually fail but appeared to the secondary database server to fail. For example, if the secondary database server did not receive responses when it signalled (pinged) the primary database server because of a slow or unstable network, it would assume that the primary database server failed and switch automatically to type standard. If the primary database server also did not receive responses when it signalled the secondary database server, it would assume the secondary database server had failed and would turn off data replication but remain in on-line mode. Now the primary and the secondary (switched to type standard) database servers are both in on-line mode.

If clients can update the data on both database servers independently, the database servers in the pair reach a state where each database server has logical-log records needed by the other. In this situation, you must start from scratch and perform initial data replication with a level-0 dbspace backup of one entire database server, as described in "Starting High-Availability Data Replication for the First Time" on page 26-11. Therefore, if your network is not entirely stable, you might not want to use automatic switchover.

What Is 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 on-line mode as a standard database server, so it can accept updates from client applications. How to perform the switchover is explained in "Changing the Type of the Secondary Database Server" on page 26-22.

Restarting After a Manual Switchover

The steps involved in restarting data replication after a manual switchover are listed in "The Secondary Database Server Is Changed to a Standard Database Server Manually" on page 26-31.

Restarting If the Secondary Database Server Is Not Switched to Standard Dynamic Server

If the secondary database server is not changed to type standard either automatically or manually, follow the steps listed in "The Secondary Database Server Was Not Changed to a Standard Database Server" on page 26-31.

Redirection and Connectivity for Data-Replication Clients

Clients connect to the database servers in a high-availability data-replication pair using the same methods with which they connect to standard database servers. These methods are explained in the descriptions of the CONNECT and DATABASE statements in the *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 amount of time, redirecting clients might not be appropriate.)

The database server does not have a transparent mechanism for directing client requests to different database servers in a high-availability data-replication pair, although you can automate this action from within the application as described in "Handling Redirection Within an Application" on page 25-30.

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 **DBPATH**
- Administrator-controlled redirection with the connectivity information
- User-controlled redirection with **INFORMIXSERVER**

The mechanism that you employ determines which CONNECT syntax you can use in your application. The following three sections describe each of the redirection mechanisms.

Automatic Redirection with DBPATH

This section explains the steps that you must follow to redirect clients with the **DBPATH** mechanism and the connectivity strategy that supports this method.

How Does the DBPATH Redirection Method Work?

The **DBPATH** redirection method relies on the fact that when an application does not explicitly specify a database server in the CONNECT statement, and the database server specified by the **INFORMIXSERVER** environment variable is unavailable, the client uses the **DBPATH** environment variable to locate the database (and database server).

So, if one of the database servers in a high-availability data-replication pair is unusable, applications that use that database server need not reset their **INFORMIXSERVER** environment variable, as long as they have their **DBPATH** environment variable 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 high-availability data-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. You *cannot* use any of the following statements for this method to work:

- 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 on **DBPATH**, refer to the *Informix Guide to SQL: Reference*.

What Does the Administrator Need to Do?

Administrators take no action to redirect clients. Administrators might need to attend to the type of the database server.

What Does the User Need 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 that are running clients must quit and restart all applications.

Administrator-Controlled Redirection with the Connectivity Information

This section explains the steps in redirecting clients with the connectivity information and the connectivity strategy that supports this method. As the following table shows, the location of the connectivity information depends on the platform.

Platform	Location of Connectivity Information
UNIX	The **INFORMIXSQLHOSTS** environment variable specifies the full pathname and filename of the connection information. The default location is **$INFORMIXDIR /etc/sqlhosts**. For more information about **INFORMIXSQLHOSTS**, see the *Informix Guide to SQL: Reference*.
Windows NT	The connectivity information is in a key in the Windows registry called **HKEY_LOCAL_MACHINE\SOFTWARE\Informix\SQLHOSTS**.

How Does the Connectivity Information-Redirection Method Work?

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 high-availability data-replication pair is unusable, an administrator can change the definition of the unavailable database server in the connectivity information. As described in "What Does the Administrator Need to Do?" the fields of the unavailable database server (except for the **dbservername** field) are changed to point to a definition of the remaining database server in the high-availability data-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 if a connection has failed and issues a reconnect statement, if necessary. If a connection has failed, redirection is handled automatically, 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*

What Does the Administrator Need to Do?

Administrators must perform the following two steps to redirect clients using the connectivity information:

1. Change the connectivity information for the clients.
2. Change other connectivity files, if necessary.

These steps are described in the following sections. For information on the connectivity information, refer to Chapter 4, "Client/Server Communications."

Changing the Connectivity Information

On the client computer, edit the connectivity information. Make the following changes:

- Comment out the entry for the failed database server.
- 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.

Figure 25-8 on page 25-28 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.

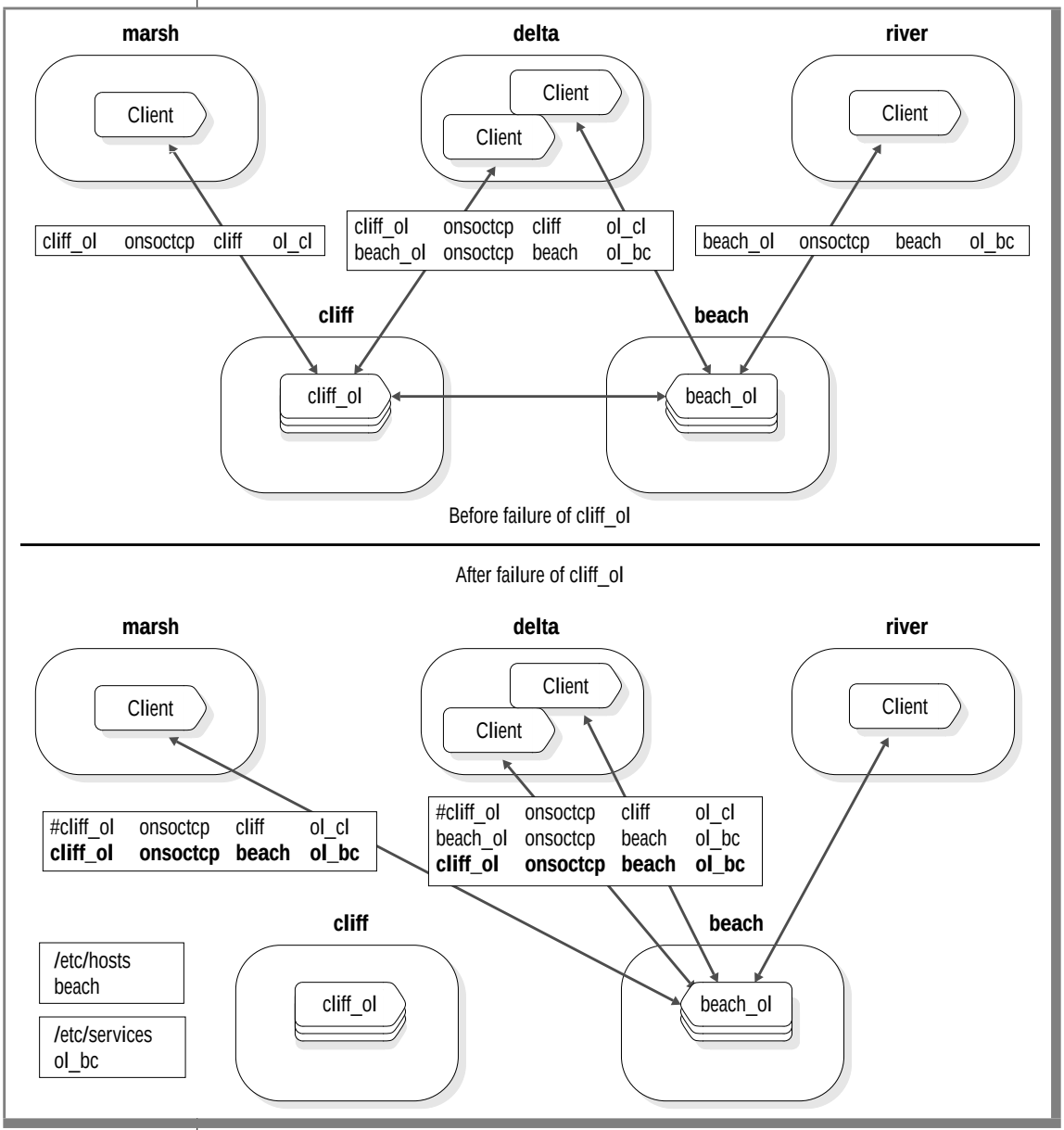
UNIX

Changing 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:

- The **/etc/hosts** file has an entry for the **hostname** of the computer that is running the database server to which you are redirecting clients.
- The **/etc/services** file has an entry for the **servicename** of the database server to which you are redirecting clients.

Figure 25-8
Connectivity Values Before and After a Failure of the cliff_ol Database Server



What Does the User Need to Do?

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.

User-Controlled Redirection with INFORMIXSERVER

This section explains the steps in redirecting clients with the **INFORMIXSERVER** environment variable and the connectivity strategy that supports that method.

How Does the INFORMIXSERVER Redirection Method Work?

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 high-availability data-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.

You can use the following connectivity statements to support this method of redirection:

- 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 Does the Administrator Need to Do?

Administrators take no action to redirect the clients. However, administrators might need to attend to the type of the database server.

What Does the User Need 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:

1. Quit their applications.
2. Change their **INFORMIXSERVER** environment variable to hold the name of the other database server in the high-availability data-replication pair.
3. Restart their applications.

Handling Redirection Within an Application

If you use the **DBPATH** or connectivity information-redirection mechanism, you can include in your clients a routine that handles errors when clients encounter a high-availability data-replication failure. The routine can call another function that contains a loop that tries repeatedly to connect with the other database server in the pair. This routine redirects clients without requiring the user to exit the application and restart it.

A Connection Loop and Database Server Type Check

Figure 25-9 shows an example of a function in a client application using the **DBPATH** redirection mechanism that loops as it attempts to reconnect. Once 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 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 “stores7”;
        if (strcmp(SQLSTATE,”00000”))
        {
            if (sqlca.sqlwarn.sqlwarn6 != ’W’)
            {
                notify_admin();
                if (tries < MAXTRIES - 1)
                    sleep(15);
                }
            else connected =1;
            }
        }
    return ((tries == MAXTRIES)? -1:0);
    }
```

Figure 25-9
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-redirection method, you might have a different connection statement, as follows:

```
EXEC SQL CONNECT TO "stores7@cliff_ol";
```

In this example, **stores7@cliff_ol** refers to a database on a database server that is recognized by the client computer. 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 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

Figure 25-10 summarizes the differences among the three redirection mechanisms.

Figure 25-10
Comparison of Redirection Methods for Different Connectivity Strategies

	DBPATH		Connectivity Information		INFORMIXSERVER
	Automatic Redirection	**User Redirection**	**Automatic Redirection**	**User Redirection**	**User Redirection**
When is a client redirected?	When the client next tries to connect with a specified database.		After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server.		When the client restarts and reads a new value for the **INFORMIXSERVER** environment variable.
Do clients need to be restarted to be redirected?	No	Yes	No	Yes	Yes
What is the scope of the redirection?	Individual clients are redirected.	Individual clients are redirected.	All clients that use a given database server are redirected.	Individual clients are redirected.	Individual clients are redirected.
Are changes to environment variables required?	No		No		Yes

Designing High-Availability Data-Replication Clients

This section discusses various design considerations (in addition to the redirection considerations discussed earlier) for clients that connect to database servers that are running data replication.

Setting Lock Mode to Wait for Access to Primary Database Server

When the database server performs a logical recovery, it normally defers index builds until the end of the recovery. However, if Dynamic Server is acting as a secondary database server, it is in logical recovery mode for as long as data replication is running. Thus, secondary database servers must use a different mechanism to perform index builds.

The mechanism used is as follows. When the secondary database server receives a logical-log record that necessitates a corresponding index build, it sends a message back to the primary database server to request a physical copy of the index. The primary database server has a lock on the table that is being updated. The owner of the lock is a **dr_btsend** thread. The application thread that is executing is free to continue processing. The **dr_btsend** thread cannot release the lock, however, until the secondary database server acknowledges receipt of the index. If the application tries to access the table while it is locked, this attempt fails unless the application has set the lock mode to wait.

Applications might see some unexpected errors if they do not have lock mode set to wait. For example, many SQL statements cause updates to the catalog table indexes. The following sequence of SQL statements fails if the lock mode of the application is not set to wait:

```
CREATE DATABASE db_name;
DATABASE db_name;
CREATE TABLE tab_name;
```

These SQL statements would fail because the CREATE DATABASE statement creates indexes on the **systables** catalog table and, therefore, places a lock on the table until the indexes are copied over to the secondary database server. Meanwhile, the CREATE TABLE statement tries to insert a row into the **systables** catalog table. The insert fails, however, because the table is locked.

This application would fail because both the CREATE DATABASE and CREATE TABLE statements cause updates to the **systables** catalog table index.

Designing Clients to Use the Secondary Database Server

To achieve a degree of load balancing when you use data replication, have some client applications use the secondary database server in a data-replication pair. Design all client applications that use the secondary database server with the following points in mind:

- Any statements that attempt to modify data fail.
- Locking and isolation levels are not the same as on the standard Dynamic Server.
- Temporary dbspaces must be used for sorting and temporary tables.

These considerations are discussed in more detail in the following sections.

No Data Modification Statements

SQL statements that update dbspaces that are in logical recovery (which includes all dbspaces on the secondary database server) are not allowed. For example, the following statements produce errors:

- ALTER FRAGMENT
- ALTER INDEX
- ALTER TABLE
- CREATE DATABASE
- CREATE INDEX
- CREATE PROCEDURE
- CREATE PROCEDURE FROM
- CREATE ROLE
- CREATE SCHEMA

- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- DELETE
- DROP DATABASE
- DROP INDEX
- DROP PROCEDURE
- DROP ROLE
- DROP SYNONYM
- DROP TABLE
- DROP TRIGGER
- DROP VIEW
- GRANT
- GRANT FRAGMENT
- INSERT
- LOAD
- RENAME COLUMN
- RENAME DATABASE
- RENAME TABLE
- REVOKE
- REVOKE FRAGMENT
- UNLOAD
- UPDATE
- UPDATE STATISTICS

To prevent clients that are using the secondary database server from issuing updating statements, you can take either of the following actions:

- Write client applications that do not issue updating statements.
- Conditionalize all updating statements.

To conditionalize statements that perform an update, make sure that client applications test **slqwarn6** of the **sqlwarn** field in the ESQL/C **sqlca** structure (and equivalent values for other SQL APIs). The database server sets **slqwarn6** to `W` when it runs as a secondary database server.

Locking and Isolation Level

Because all clients that use the secondary database server only read data, locking to ensure isolation between those clients is not required. However, a client that uses the secondary database server is not protected from the activity of users on the primary database server because the **logrecvr** threads that perform logical recovery do not use locking.

For example, if a client connected to the secondary database server reads a row, nothing prevents a user on the primary database server from updating that row, even if the client connected to the secondary database server has issued a SET ISOLATION TO REPEATABLE READ statement. The update is reflected on the secondary database server as the logical-log records for the committed transaction are processed. Thus, all queries on the secondary database server are essentially dirty with respect to changes that occur on the primary database server, even though a client that uses the secondary database server might explicitly set the isolation level to something other than Dirty Read.

Using Temporary Dbspaces for Sorting and Temporary Tables

Even though the secondary database server is in read-only mode, it still does writing when it needs to perform a sort or create a temporary table. "What Is a Temporary Dbspace?" on page 13-20 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 one (or all) of the following actions:

- Ensure that a temporary dbspace exists. For instructions on creating a temporary dbspace, see "Creating a Dbspace" on page 14-11.
- Set the DBSPACETEMP parameter in the ONCONFIG file of the secondary database server to the temporary dbspace or spaces.
- Have clients that connect to the secondary database server and need to take advantage of that temporary dbspace set their **DBSPACETEMP** environment variable to the name of that dbspace or spaces.

Chapter

26

Using High-Availability Data Replication

Planning for High-Availability Data Replication 26-4

Configuring High-Availability Data Replication 26-5
- Meeting Hardware and Operating-System Requirements 26-5
- Meeting Database and Data Requirements 26-6
- Meeting Database Server Configuration Requirements 26-6
  - Version 26-7
  - Dbspace and Chunk Configuration 26-7
  - Mirroring 26-7
  - Physical-Log Configuration 26-8
  - Dbspace and Logical-Log Tape Backup Devices 26-8
  - Logical-Log Configuration 26-8
  - Shared-Memory Configuration 26-9
  - High-Availability Data-Replication Parameters 26-9
- Configuring High-Availability Data-Replication Connectivity 26-9

Starting High-Availability Data Replication for the First Time 26-11

Performing Basic Administration Tasks 26-15
- Changing Database Server Configuration Parameters 26-15
- Dbspace and Logical-Log File Backups 26-16
- Changing the Logging Status of Databases 26-16
- Adding and Dropping Chunks, Dbspaces, and Blobspaces 26-17
- Using and Changing Mirroring of Chunks 26-17
- Managing the Physical Log. 26-18
- Managing the Logical Log 26-19
- Managing Virtual Processors 26-19
- Managing Shared Memory 26-19

Changing the Database Server Mode 26-20

Changing the Database Server Type 26-21
Changing the Type of the Primary Database Server 26-22
Changing the Type of the Secondary Database Server. 26-22

Restoring Data If Media Failure Occurs 26-23
Restoring After Media Failure on the Primary Database Server 26-23
Restoring After Media Failure on the Secondary Database Server 26-24

Restarting High-Availability Data Replication After a Failure 26-26
Restarting After Critical Data Is Damaged 26-26
Critical Media Failure on the Primary Database Server 26-26
Critical Media Failure on the Secondary Database Server . . . 26-28
Critical Media Failure on Both Database Servers 26-28
Restarting If Critical Data Is Not Damaged 26-28
Restarting After a Network Failure. 26-29
Restarting If the Secondary Database Server Fails. 26-29
Restarting If the Primary Database Server Fails 26-31

This chapter describes how to use high-availability data replication. If you plan to use high-availability data replication, read this entire chapter first. If you plan to use Informix Enterprise Replication, see the *Guide to Informix Enterprise Replication*. This chapter covers the following topics:

- Planning for high-availability data replication
- Configuring a system for high-availability data replication
- Starting high-availability data replication
- Operating database servers that use high-availability data replication
- Managing the mode of a database server in a high-availability data-replication pair
- Changing the type of a database server in a high-availability data-replication pair
- Restoring data after a media failure
- Managing high-availability data replication after a failure

A companion chapter, Chapter 25, "What Is High-Availability Data Replication?" explains what high-availability data replication is, how it works, and how to design client applications for a high-availability data-replication environment.

Planning for High-Availability Data Replication

Before you start setting up computers and database servers to use high-availability data replication, 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 wish to replicate, migrate and redistribute this data so that it can be managed by a single database server.
- Ensure that all databases that you want to replicate use transaction logging. To turn on transaction logging, see Chapter 17, "Managing Database-Logging Status."
- Develop client applications to make use of both database servers in the high-availability data-replication pair. For a discussion of design considerations, refer to "Redirection and Connectivity for Data-Replication Clients" on page 25-22 and "Designing Clients to Use the Secondary Database Server" on page 25-35.
- Create a schedule for starting high-availability data replication for the first time.
- Design a dbspace 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 high-availability data replication after a failure. Read "Redirection and Connectivity for Data-Replication Clients" on page 25-22.

Configuring High-Availability Data Replication

To configure your system for high-availability data replication, you must take the following actions:

- Meet hardware and operating-system requirements
- Meet database and data requirements
- Meet database server configuration requirements
- Configure high-availability data-replication connectivity

Each of these topics is explained in this section.

Meeting Hardware and Operating-System Requirements

For a high-availability high-availability data-replication database server pair to function, it must meet the following hardware requirements:

- The computers that run the primary and secondary database servers must be identical (same vendor and architecture).
- The operating systems on the computers that run the primary and secondary database servers must be identical.
- The hardware that runs the primary and secondary database servers must support network capabilities.
- The amount of disk space allocated to nontemporary 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.

Meeting Database and Data Requirements

For a high-availability data-replication database server pair to function, you must meet the following database and data requirements:

- 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 high-availability data replication, see either "Turning On Transaction Logging with ON-Archive" on page 17-5 or "Turning On Transaction Logging with ontape" on page 17-7.

- If your primary database server has TEXT or BYTE objects stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal high-availability data-replication processing. TEXT or BYTE data within dbspaces *is* replicated, however.

Meeting Database Server Configuration Requirements

For a high-availability data-replication database server pair to function, you must meet the following database server configuration requirements.

To meet these requirements, you must fully configure each of the database servers. For information on configuring a database server, refer to "Configuring the Database Server" on page 3-7. You can then use the relevant aspects of that configuration to configure the other database server in the pair.

Version

The versions of the database server on the primary and secondary database servers must be identical.

Dbspace and Chunk Configuration

The number of dbspaces, the number of chunks, their sizes, their pathnames, and their offsets must be identical on the primary and secondary database servers.

The configuration must contain at least one temporary dbspace. See "Using Temporary Dbspaces for Sorting and Temporary Tables" on page 25-38.

You can use symbolic links for the chunk pathnames, as explained in "Creating a Link to Each Raw Device on UNIX" on page 14-9.

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME (see page 33-76)
- ROOTOFFSET (see page 33-76)
- ROOTPATH (see page 33-77)
- ROOTSIZE (see page 33-78)

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. If you specify a mirrored chunk for the root chunk of the primary database server, however, you must also specify a mirrored 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 (see page 33-54)
- MIRRORPATH (see page 33-54)

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:

- PHYSDBS (see page 33-70)
- PHYSFILE (see page 33-71)

Dbspace and Logical-Log Tape Backup Devices

You can specify different tape devices for the primary and secondary database servers.

The tape size and tape block size for the dbspace and logical-log tape backup devices should be identical. The following ONCONFIG parameters must have the same value on each database server:

- LTAPEBLK (see page 33-47)
- LTAPESIZE (see page 33-49)
- TAPEBLK (see page 33-87)
- TAPESIZE (see page 33-90)

Logical-Log Configuration

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:

- LOGFILES (see page 33-42)
- LOGSIZE (see page 33-43)

Shared-Memory Configuration

Set all the shared-memory configuration parameters to the same values on the two database servers.

High-Availability Data-Replication Parameters

The following parameters are specific to high-availability data replication and must be set to the same value on both database servers in the high-availability data-replication pair:

- DRAUTO (see page 33-22)
- DRINTERVAL (see page 33-23)
- DRLOSTFOUND (see page 33-24)
- DRTIMEOUT (see page 33-25)

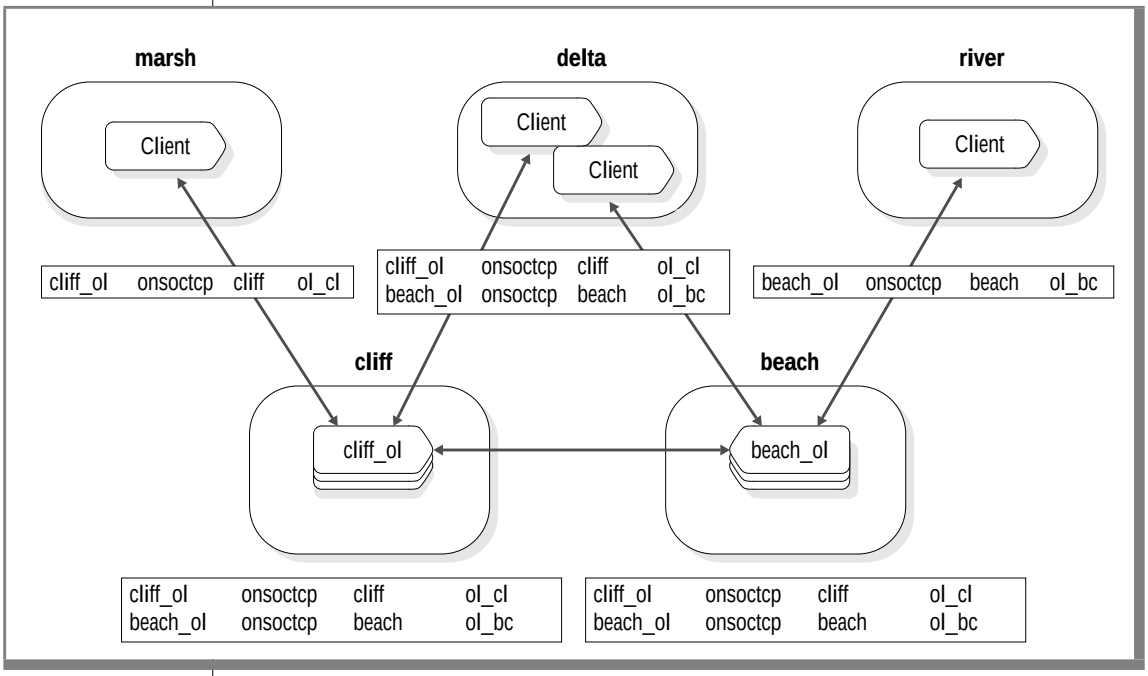
Configuring High-Availability Data-Replication Connectivity

For a high-availability data-replication database server pair to function, the database servers in the high-availability data-replication pair 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 high-availability data-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

Figure 26-1 shows a sample high-availability data-replication configuration and example connectivity values necessary for high-availability data replication.

Figure 26-1
Example Connectivity Values for Database Servers in a High-Availability Data-Replication Pair



UNIX

In addition to connectivity information, the computers that are running Dynamic Server in a high-availability data-replication pair must have entries for the other computer and service in their **/etc/hosts** and **/etc/services** files. ♦

Starting High-Availability Data Replication for the First Time

After you complete the high-availability data-replication configuration, you are ready to start high-availability data replication. This section describes the necessary steps for starting high-availability data replication.

Suppose you wish to start high-availability data replication on two database servers, **ServerA** and **ServerB**. The procedure for starting high-availability data replication, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps. Figure 26-2 on page 26-13 lists the commands required to perform each step. You can perform some of the steps using either the ON-Archive or the **ontape** utility. In such cases, the ON-Archive command and the equivalent **ontape** command are both indicated. You must employ the same utility throughout the procedure, however. Figure 26-2 also shows messages sent to the message log.

To start high-availability data replication

1. Create a level-0 dbspace backup of **ServerA**.
2. 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 high-availability data-replication connection with the other database server in the high-availability data-replication pair and to start high-availability data-replication 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 on-line and is not set to type secondary, so the high-availability data-replication connection is not established.
3. Perform a physical restore of **ServerB** from the level-0 dbspace backup that you created in step 1. Do not perform a logical restore. If you are using the **ontape** utility for your archiving tasks, use the **ontape -p** option. You cannot use the **ontape -r** option because it performs both a physical and a logical restore.

4. Use the **onmode -d** command to set the type of **ServerB** to secondary and indicate the associated primary database server. **ServerB** tries to establish a high-availability data-replication connection with the primary database server (**ServerA**) and start operation. The connection should be successfully established.

   Before high-availability data replication begins, the secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 1. 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 5.

5. 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.

Figure 26-2
Steps to Start High-Availability Data Replication for the First Time

Step	On the Primary	On the Secondary

1	**ON-Archive command**
	`Onarchive> ARCHIVE/DBSPACESET=*`
	ontape command
	`% ontape -s`
	Messages to message log
	`Level 0 archive started on rootdbs.`
	`Archive on rootdbs completed.`	

2	**onmode command**
	`%onmode -d primary sec_name`
	Messages to message log
	`DR: new type = primary, secondary server name = sec_name`
	`DR: trying to connect to secondary server`
	`DR: Cannot connect to secondary server`	

3		**ON-Archive command**
		`ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST=rid/TAPE=(primary:/dev/remotedrive`
		ontape command
		`% ontape -p`
		Answer `no` when prompted to back up the logs.
		Messages to message log
		`Informix Dynamic Server Initialized -- Shared Memory Initialized`
		`Recovery Mode`
		`Physical restore of rootdbs started.`
		`Physical restore of rootdbs completed.`

(1 of 3)

Step	On the Primary	On the Secondary
4		**onmode command** `% onmode -d secondary` *`prim_name`* **Messages to message log** `DR: new type = secondary, primary 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 5 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.`
		`Logical Recovery allocating n worker threads ('OFF_RECVRY_THREADS').`
		`Logical Recovery Started`
		`Start Logical Recovery - Start Log n, End Log?`
		`Starting Log Position - n 0xnnnnn`
		`DR: Secondary server operational`

(2 of 3)

Step	On the Primary	On the Secondary

5		**ON-Archive command**
		`ONDATARTR> RETRIEVE/LOGFILE/TAPE= (primary:/dev/remotedevice)`
		ontape command
		`% ontape -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.`
		`Logical Recovery allocating n worker threads ('OFF_RECVRY_THREADS').`
		`Logical Recovery Started`
		`Start Logical Recovery - Start Log n, End Log?`
		`Starting Log Position - n 0xnnnnn`
		`DR: Secondary server operational`

(3 of 3)

Performing Basic Administration Tasks

This section contains instructions on how to perform basic Dynamic Server administration tasks once your system is running high-availability data replication.

Changing Database Server Configuration Parameters

Some of the Dynamic Server configuration parameters must be set to the same value on both database servers in the high-availability data-replication pair (as listed under "Meeting Database Server Configuration Requirements" on page 26-6). Other Dynamic Server configuration parameters can be set to different values.

If you need to change a configuration parameter that must have the same value on both database servers, you must change the value of that parameter in the ONCONFIG file of both database servers. To make changes to ONCONFIG files, perform the following steps:

1. Bring each database server off-line 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. Bring each database server back on-line. Start with the last database server that you brought off-line. For example, if you brought the secondary database server off-line last, bring the secondary database server on-line first. Figure 26-1 on page 26-10 and Figure 26-2 on page 26-13 list the procedures for bringing the primary and secondary database servers back on-line.

If the configuration parameter does not need to have the same value on each database server in the high-availability data-replication pair, you can change the value on the primary or secondary database server individually.

Dbspace and Logical-Log File Backups

When you use high-availability data replication, you must back up logical-log files and create dbspace backups of your data, just as you would with a standard Dynamic Server database server. You need to perform dbspace and logical-log file backups only on the primary database server. Be prepared, however, to perform dbspace 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 dbspace backups and logical-log backups) must be identical on the primary and secondary database servers.

Changing the Logging Status of Databases

You cannot add transaction logging to databases on the primary database server while you are using high-availability data replication. You can turn logging off for a database; however, subsequent changes to that database are not duplicated on the secondary database server.

If you must add logging to a database, you can turn high-availability data replication off, add logging, and then perform a dbspace backup and restore as described in "Starting High-Availability Data Replication for the First Time" on page 26-11.

Adding and Dropping Chunks, Dbspaces, and Blobspaces

You can perform disk-layout operations, such as adding or dropping chunks, dbspaces, and blobspaces, 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 high-availability data-replication pair remains consistent.

Because the directory pathname or the actual file for chunks must exist before you create them, make sure the pathnames (and offsets, if applicable) exist on the secondary database server before you create a chunk on the primary database server.

Using and Changing Mirroring of Chunks

You do not have to set the MIRROR configuration parameter to the same value on both database servers in the high-availability data-replication pair. In other words, you can enable or disable mirroring on either the primary or the secondary database server independently.

Before you can add a mirrored chunk, the disk space for that chunk must already be allocated on both the primary and secondary database servers. For general information on allocating disk space, see "Allocating Disk Space" on page 14-4.

If you want to mirror a dbspace on one of the database servers in the high-availability data-replication pair, you must create mirrored chunks for that dbspace on *both* database servers. For general information on allocating disk space, see "Allocating Disk Space" on page 14-4.

You can perform disk-layout operations only from the primary database server. Thus, you can add or drop a mirrored chunk only from the primary database server. A mirrored 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 manually perform mirror recovery for the newly added mirror chunk on the secondary database server to bring it to online mode. For more information, see "Recovering a Mirrored Chunk" on page 24-11.

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. You can turn mirroring off for a dbspace on the secondary database server independent of the primary server using **onspaces -r**. For more information on turning mirroring off, see "Ending Mirroring" on page 24-12.

You can take down a mirrored chunk or recover a mirrored chunk on either the primary or secondary database server. These processes are transparent to high-availability data replication.

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; however, the PHYSDBS and PHYSFILE parameters in the secondary ONCONFIG file are not updated. You must change these parameters manually by editing the ONCONFIG file. For the procedure to follow for making this change, see "Changing Database Server Configuration Parameters" on page 26-15.

For information on changing the size and location of the physical log, refer to Chapter 21, "Managing the Physical Log."

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 19, "Managing Logical-Log Files." 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 dbspace 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 dbspace 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 to follow for making this change, see "Changing Database Server Configuration Parameters" on page 26-15.

Changing the Database Server Mode

The effects of changing the mode of a database server in a high-availability data-replication pair differ depending on whether you are changing the mode of the primary or the secondary database server.

Figure 26-3 summarizes the effects of changing the mode of the primary database server.

Figure 26-3
Mode Changes on the Primary Database Server

On the Primary	On the Secondary	To Restart High-Availability Data Replication
Any mode → off-line (**onmode -k**)	Secondary receives errors. High-availability data replication is turned off. If DRAUTO is set to 0 (OFF), mode remains read-only. If DRAUTO is set to 1 (RETAIN_TYPE) or 2 (REVERSE_TYPE), secondary switches to standard type and can accept updates.	Treat it like a failure of the primary. Three different scenarios are possible, depending on what you do with the secondary while the primary is off-line: ■ "The Secondary Database Server Was Not Changed to a Standard Database Server" on page 26-31 ■ "The Secondary Database Server Is Changed to a Standard Database Server Manually" on page 26-31 ■ "The Secondary Database Server Is Changed to a Standard Database Server Automatically" on page 26-33
On-line → quiescent (**onmode-s** / **onmode-u**)	Secondary does not receive errors. High-availability data replication remains on. Mode remains read-only.	Use **onmode -m** on the primary.

Figure 26-4 summarizes the effects of changing the mode of the secondary database server.

Figure 26-4
Mode Changes on the Secondary Database Server

On the Secondary	On the Primary	To Restart High-Availability Data Replication
Read-only → off-line (**onmode -k**)	Primary receives errors. High-availability data replication is turned off.	Treat it like a failure of the secondary. Follow the procedures in "Restarting If the Secondary Database Server Fails" on page 26-29.

Changing the Database Server Type

You might want to stop the high-availability data-replication process manually by changing the type of the database server to *standard*. The effects of this change are different from changing the mode of a database server (described in "Changing the Database Server Mode" on page 26-20). When you take the now standard database server off-line and bring it back on-line, it does not attempt to connect to the other database server in the high-availability data-replication pair.

The utility that you use to change the database server type is **onmode**. Reference information for **onmode** is in "onmode: Mode and Shared-Memory Changes" on page 35-28.

You can change the type of either the primary or the secondary database server. When you change the database server type to *standard*, the type of the other database server in the high-availability data-replication pair does not change, but high-availability data replication is turned off.

Changing the Type of the Primary Database Server

The primary database server can be in on-line mode when you change its type to standard.

Execute the following command from the operating-system prompt of the computer that is running the primary database server:

```
% onmode -d standard
```

This command stops high-availability data replication and leaves the database server in on-line mode. If DRAUTO is set to 0, the secondary database server remains in read-only mode and cannot accept updates from clients (because its type is still secondary). If DRAUTO is set to 1 (RETAIN_TYPE) or 2 (REVERSE_TYPE), the secondary database server switches to type standard. In either case, high-availability data replication is turned off on the secondary database server.

To change the database server back to type primary and restart high-availability data replication, execute the following command:

```
% onmode -d primary secondary
```

Changing the Type of the Secondary Database Server

Execute the following command from the operating-system prompt of the computer that is running the secondary database server:

```
% onmode -d standard
```

Once you change the secondary database server to a standard database server, applications can update the data managed by that database server. If you later decide to change the type of the database server back to type secondary and restart high-availability data replication, you must follow the entire procedure in "Starting High-Availability Data Replication for the First Time" on page 26-11.

Restoring Data If Media Failure Occurs

The result of 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 Media Failure on the Primary Database Server

Figure 26-5 summarizes the various scenarios for restoring data if the primary database server suffers media failure. The following issues are relevant:

1. If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.
2. 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 does contain critical media, the primary database server fails. You have to do 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 26-26.

   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 dbspace backup and next a logical restore of all logical-log records written since that dbspace backup. For more information on performing a warm restore using ON-Bar, see your *Backup and Restore Guide*. For more information on performing a warm restore using ON-Archive, see your *Archive and Backup Guide*. You must back up all logical-log files before you perform the warm restore.

Figure 26-5
Scenarios for Media Failure on the Primary Database Server

High-Availability Data-Replication 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 26-26.
Primary	Yes	Yes	Primary database server remains on-line. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-11.
Primary	No	No	Primary database server remains on-line. 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 on-line. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-11.

Restoring After Media Failure on the Secondary Database Server

Figure 26-6 summarizes the various scenarios for restoring data if the secondary database server suffers media failure. The following issues are relevant:

1. If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.
2. In cases where the chunks are not mirrored, the secondary database server fails if the disk contains critical media but remains on-line if the disk does not contain critical media. In both cases, you have to do a full restore using the dbspace backups on the primary database server. (See "Restarting After Critical Data Is Damaged" on page 26-26.) 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 do a full restore.

Figure 26-6
Scenarios for Media Failure on the Secondary Database Server

High-Availability Data-Replication Server	Critical Media?	Chunks Mirrored?	Effect of Failure
Secondary	Yes	No	Secondary database server fails. Primary database server receives errors. High-availability data replication is turned off. Follow the procedure in "Restarting After Critical Data Is Damaged" on page 26-26
Secondary	Yes	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-11.
Secondary	No	No	Secondary database server remains on-line in read-only mode. Primary database server receives errors. High-availability data replication is turned off. Follow the procedure in "Restarting After Critical Data Is Damaged" on page 26-26.
Secondary	No	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-11.

Restarting High-Availability Data Replication After a Failure

"What Are High-Availability Data-Replication Failures?" on page 25-15 discusses the various types of high-availability data-replication failure. The procedure that you must follow to restart high-availability data replication depends on whether 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 high-availability data replication 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

To restart high-availability data replication after the primary database server suffers a critical media failure, perform the following steps. Figure 26-7 lists the commands required to perform this procedure:

1. If the original secondary database server was changed to a standard database server manually (DRAUTO = 0), bring this database server to quiescent mode and then use the **onmode -d** command to change the type back to secondary.

   If DRAUTO = 1, 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 on-line.

   If DRAUTO = 2, the secondary database server becomes a primary as soon as the connection ends when the old primary database server fails rather than when the old primary is restarted.

   In the following steps, it is assumed that DRAUTO is set to 0 or to 1.
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 high-availability data replication. 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, high-availability data replication 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.

Figure 26-7
Steps for Restarting High-Availability Data Replication 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	**ontape command**
	`% ontape -p`
	ON-Archive command
	`ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST=rid/TAPE=(primary:/dev/remotedrive)`	

3	**onmode command** `% onmode -d primary` *`sec_name`*	

4	**ontape command**
	`% ontape -l`
	ON-Archive command
	`ONDATARTR> RETRIEVE/LOGFILE/TAPE=(secondary:/dev/remotedevice)`	

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 High-Availability Data Replication for the First Time" on page 26-11.

Critical Media Failure on Both Database Servers

In the unfortunate event that both of the computers that are running database servers in a high-availability data-replication pair experience a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, perform the following tasks to restart high-availability data replication:

1. Restore one database server—it does not matter which one—from dbspace and logical-log backup tapes.
2. After you restore one database server, treat the other failed database server as if it has no data on the disks, and you are starting high-availability data replication for the first time. (See "Starting High-Availability Data Replication for the First Time" on page 26-11). Use the functioning database server with the intact disk(s) 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 five scenarios, each requiring different procedures for restarting high-availability data replication, 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 manually (DRAUTO = 0).
- The primary database server fails, and the secondary database server is changed to a standard database server automatically (DRAUTO = 1 or DRAUTO = 2).

Restarting After a Network Failure

After a network failure with DRAUTO set to 0 (OFF), the primary database server is in on-line mode, and the secondary database server is in read-only mode. High-availability data replication is turned off on both database servers (state = off). When the connection is reestablished, you can restart high-availability data replication by issuing **onmode -d secondary** *primary_name* on the secondary database server. Restarting high-availability data replication 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.

If DRAUTO is set to 1 (RETAIN_TYPE) or 2 (REVERSE_TYPE), the procedure described in the preceding paragraph does not work because the type of the secondary database server has changed to standard. If a network failure occurs, and you are using automatic switchover, there is a risk that both database servers will be updated independently. (See "Using Automatic Switchover Without a Reliable Network" on page 25-21.) In this case, you must follow the procedure discussed in "Starting High-Availability Data Replication for the First Time" on page 26-11.

Restarting If the Secondary Database Server Fails

If you need to restart high-availability data replication after a failure of the secondary database server, complete the steps in Figure 26-8. 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.

Figure 26-8
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 on-line mode.	`% oninit`
		If you receive the following message in the message log, continue with step 2:
		`DR: Start Failure recovery from tape`

2		**ON-Archive command**
		`Onarchive>`
		`CATALOG/VSET=remote_logs/VOLUME=volnum/SID=sysid`
		`Onarchive>RETRIEVE/LOGFILE/VSET=remote_logs`
		ontape command
		`% ontape -l`

Restarting If the Primary Database Server Fails

The following sections describe how to restart high-availability data replication 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 high-availability data replication after a failure of the primary database server if the secondary database server is not changed to a standard database server, simply bring the primary database server back online using **oninit**.

The Secondary Database Server Is Changed to a Standard Database Server Manually

If you need to restart high-availability data replication after a failure of the primary database server, and you have manually changed the secondary database server to be a standard database server, complete the steps in Figure 26-9.

Figure 26-9
Steps to Restart If You Manually Changed the Secondary Database Server to a Standard Database Server

Step	On the Primary Database Server	On the Secondary Database Server
1		`% onmode -s` This step takes the secondary database server (now a standard) to quiescent mode. All clients that are connected to this database server will have to disconnect. Applications that perform updates must be redirected to the primary. See "Redirection and Connectivity for Data-Replication Clients" on page 25-22.
2		`% onmode -d secondary` *`prim_name`*

(1 of 2)

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 logical-log files that you have backed up and freed are 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. For ON-Archive users: In the next step, be sure to use a vset with the device type defined to be a device on the secondary database server.	

4	If you are prompted to recover logical-log records from tape, perform this step.
	ON-Archive command
	`Onarchive>`
	`CATALOG/VSET=remote_logs/VOLUME= volnum/SID=sysid`
	`Onarchive>RETRIEVE/LOGFILE/VSET= remote_logs`
	ontape command
	`% ontape -l`	

(2 of 2)

The Secondary Database Server Is Changed to a Standard Database Server Automatically

If you need to restart high-availability data replication after a failure of the primary database server, and the secondary database server was automatically changed to a standard database server (as described in "What Is Automatic Switchover?" on page 25-19), complete the steps in Figure 26-10.

Figure 26-10
Steps to Restart If You Automatically Changed the Secondary Database Server to a Standard Database Server

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 initialized. 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, 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. For ON-Archive users: In the next step, be sure to use a vset with the device type defined to be a device on 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 25-22. If DRAUTO = 2, the secondary database server switches to primary and then standard automatically. The old primary becomes a secondary after it restarts and connects to the other server and determines that it is now a primary.

(1 of 2)

Step	On the Primary Database Server	On the Secondary Database Server

2	If you are prompted to recover logical-log records from tape, perform this step.
	`ON-Archive command`
	`Onarchive>`
	`CATALOG/VSET=remote_logs/VOLUME=volnum/SID=sysid`
	`Onarchive>RETRIEVE/LOGFILE/VSET=remote_logs`
	ontape command
	`% ontape -l`	

(2 of 2)

Chapter

27

What Is Consistency Checking?

Performing Periodic Consistency Checking 27-4
Verify Consistency. 27-4
oncheck -cr 27-5
oncheck -cc 27-5
oncheck -ce 27-6
oncheck -cI 27-6
oncheck -cD 27-6
Monitor for Data Inconsistency 27-7
Retain Consistent Level-0 Dbspace 27-8

Dealing with Corruption 27-9
Symptoms of Corruption 27-9
Run oncheck First 27-9
I/O Errors on a Chunk 27-10

Collecting Diagnostic Information 27-11

Informix Dynamic Server is 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 into the database server message log. The database server also collects further diagnostics information in a separate file that might be useful to 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

Performing Periodic Consistency Checking

To gain the maximum benefit from consistency checking and to ensure the integrity of dbspace backups, Informix recommends that you 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.

Each of these actions is described in the following sections.

Verify Consistency

Because of the time needed for this check and the possible contention that the checks can cause, schedule this check for times when activity is at its lowest. Informix recommends that you perform this check just prior to creating a level-0 dbspace backup.

Run the following commands as part of the consistency check:

- **oncheck -cr**
- **oncheck -cc**
- **oncheck -ce**
- **oncheck -cI** *dbname*
- **oncheck -cD** *dbname*

The following sections describe these commands.

You can run each of these commands while the database server is in on-line mode. For information on how **oncheck** locks objects as it checks them and which users can run **oncheck,** see "Locking and oncheck" on page 35-7.

In most cases, if one or more of these checks 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.

If you use ON-Bar for your backup and restore tool, see your *Backup and Restore Guide* for information on how to restore your data from a dbspace backup. If you use ON-Archive or **ontape** for your backup and restore tool, see your *Archive and Backup Guide*.

oncheck -cr

Execute **oncheck -cr** to validate the database server reserved pages that reside at the beginning of the initial chunk of the root dbspace. These pages contain the primary the database server overhead information. If this command detects errors, perform a data restore from dbspace backup.

This command might report warnings. In most cases, these warnings call your attention to situations of which you are already aware.

oncheck -cc

Execute **oncheck -cc** to validate the system catalog tables for each of the databases that the database server manages. Each database contains its own system catalog, which contains information on the database tables, columns, indexes, views, constraints, stored procedures, and privileges.

If a warning appears after you execute **oncheck -cc**, 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 with your database design. For example, the following warning might appear if you execute **oncheck -cc** on a database that has no synonyms defined for any table:

```
WARNING: No syssyntable records found.
```

This message indicates only that no synonym exists for any table; that is, the system catalog contains no records in the table **syssyntable**.

However, if **oncheck -cc** returns an error message, the situation is quite different. Contact Informix Technical Support immediately.

oncheck -ce

Execute **oncheck -ce** to validate the extents in every database. Extents must not overlap. If this command detects errors, perform a data restore from a dbspace backup. If this command detects errors, perform a data restore from a dbspace backup.

oncheck -cI

Execute **oncheck -cI** for each database to validate indexes on each of the tables in the database. If this command detects errors, drop and re-create the affected index.

oncheck -cD

Execute **oncheck -cD** to validate the pages for each of the tables in the database. If this command 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 *Informix Migration Guide*. If this procedure does not succeed, perform a data restore from dbspace backup.

Monitor 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 "What Is the Message Log?" on page 29-7.)

Figure 27-1 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 27-1
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***
- **/*pathname*/core** ♦

UNIX

In all cases, ***xxx*** will be 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 specified by the ONCONFIG parameter DUMPDIR.

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`.

UNIX

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 *pathname* for the **core** file is the directory from which the database server was last invoked. ♦

Most of the general assertion-failure messages are followed by additional information that usually includes the tblspace where the error was detected. If this information is available, run **oncheck -cD** on the database or table. 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 will almost always be either -105 or -172, as follows:

```
-105 ISAM error: bad isam file format
-172 ISAM error: Unexpected internal error
```

Chapter 36, "Message-Log Messages," provides additional details about the objectives and contents of messages.

Retain Consistent Level-0 Dbspace

After you perform the checks described in "Verify Consistency" on page 27-4 without errors, create a level-0 dbspace backup. Retain this dbspace backup and all subsequent logical-log backup tapes until you complete the next consistency check. Informix recommends that you perform the consistency checks before every level-0 dbspace backup. However, if you do not, then at minimum, keep all the tapes necessary to recover from the dbspace 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.

Symptoms of Corruption

The database server alerts the user and administrator to possible corruption through the following means:

- 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 "Monitor for Data Inconsistency" on page 27-7.
- The **oncheck** utility returns errors.

Run oncheck First

At the first indication of corruption, run **oncheck -cI** to determine if corruption exists in the index. If you run **oncheck -cI** while the database server is in on-line 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 on-line mode (the database server locks the table and index). If you run **oncheck -cI** in quiescent mode, and corruption is detected, **oncheck** prompts you to confirm whether the utility should attempt to repair the corruption.

If **oncheck** reports bad key information in an index, drop the index and re-create it. If **oncheck** cannot find or access the table or database, perform the checks described in "Verify Consistency" on page 27-4.

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 mirrored chunk. A message written to the database server message log contains the name of the I/O performed and an operating-system error number that identifies the cause of the I/O error.

If the down chunk is mirrored, the database server continues to operate using the mirrored 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 mirrored chunk data. For information on how to recover a mirrored chunk, see "Recovering a Mirrored Chunk" on page 24-11.

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates an abort. 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 dbspace backup.

If you take the database server to off-line mode when a chunk is marked as down (D), you can reinitialize 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 Informix Technical Support whenever one occurs. The diagnostic information collected is intended for the use of 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 Informix Technical Support indicates that you can do so. Send a fax or email with the **af**.***xxx*** file to Informix Technical Support. This 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 abort sequence. The following ONCONFIG parameters are on UNIX and Windows NT:

- DUMPDIR page 33-34
- DUMPSHMEM page 33-35

UNIX

The following ONCONFIG parameters are on UNIX only:

- DUMPCNT page 33-32
- DUMPCORE page 33-33
- DUMPGCORE page 33-34

♦

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.

Chapter

28

Situations to Avoid

Situations to Avoid in Administering the Database Server 28-3

Occasionally, database server administrators think of a shortcut that seems like a good idea. Because of the complexity of the database server, an idea that appears to be an efficient time saver can create problems elsewhere during operation. This chapter tries to protect you from bad ideas that sound good.

Your *Backup and Restore Guide* provides advice on situations to avoid when using ON-Bar to back up or restore database server data. Your *Archive and Backup Guide* provides advice on situations to avoid when you use ON-Archive or **ontape** to backup or restore data.

Situations to Avoid in Administering the Database Server

The following ideas might sound good in theory, but they have unexpected consequences that could adversely affect your database server performance:

- Never make changes to the tables in the **sysmaster** database.
- Never kill a database server process or thread (virtual processor). When you kill a database server process, the database server terminates.
- Limit the number of CPU virtual processors to the number of CPUs that are in your hardware configuration.
- Never bring on-line two different types of network services with the same service name. For information on service names, see "The sqlhosts File or Registry" on page 4-27.
- Avoid transactions that span a significant percentage of available logical-log space. See the descriptions of the LTXHWM and LTXEHWM parameters in Chapter 33, "Configuration Parameters."

UNIX

- Do not rely on **dbexport** (a utility that creates a copy of your database for migrating) as an alternative to creating routine dbspace backups. ♦
- Do not run utilities that send output to tape in background mode.
- Before you move a chunk from one dbspace or blobspace to another, perform a level-0 dbspace backup of both spaces (that is, the *before* space and the *after* space). If you do not perform a level-0 dbspace backup, a potential problem exists if the two spaces involved are restored in parallel.
- Do not locate mirrored chunks on the same device as the primary chunks. Ideally, place the mirrored chunks on devices that are managed by a different controller than the one that manages the primary chunks.

Monitoring

Section VII

Chapter

29

Monitoring the Database Server

Information That You Can Monitor 29-5

Sources of Information for Monitoring the Database Server 29-6
What Is the Message Log? 29-7
Why Read the Message Log? 29-7
Changing the Destination for Message-Log Messages 29-7
Monitoring the Message Log 29-7
Event Alarm 29-8
What Is the Console? 29-12
Monitoring with ON-Monitor 29-12
Monitoring with SMI Tables 29-12
Monitoring with onstat and oncheck Utilities 29-12
Monitoring with onperf 29-13
Monitoring with the onstat Banner Line 29-13

Monitoring Configuration Information 29-14
Using Command-Line Utilities 29-14
Using ON-Monitor 29-15

Monitoring Checkpoint Information 29-16
Using Command-Line Utilities 29-16
Using ON-Monitor 29-17
Using SMI Tables 29-17

Monitoring Shared Memory 29-18
Monitoring Shared-Memory Segments 29-18
Monitoring Shared-Memory Profile 29-19
Using Command-Line Utilities 29-19
Using ON-Monitor 29-19
Using SMI Tables 29-20

Monitoring Buffers 29-20
Using Command-Line Utilities 29-20
Using ON-Monitor 29-23
Using SMI Tables 29-23
Monitoring Buffer-Pool Activity 29-23
Using Command-Line Utilities 29-24
Using SMI Tables 29-26
Monitoring Latches 29-27
Using Command-Line Utilities 29-27
Using ON-Monitor 29-28
Using SMI Tables 29-28
Monitoring Locks 29-29
Using Command-Line Utilities 29-29
Using ON-Monitor 29-30
Using SMI Tables 29-31

Monitoring Active Tblspaces 29-32
Using Command-Line Utilities 29-32

Monitoring Virtual Processors 29-33
Using Command-Line Utilities 29-33
Using SMI Tables 29-35

Monitoring Sessions and Threads 29-35
Using Command-Line Utilities 29-35
Using ON-Monitor 29-39
Using SMI Tables 29-39

Monitoring PDQ Resources and Queries 29-40

Monitoring Transactions 29-41
Using Command-Line Utilities 29-41

Monitoring Databases 29-43
Using ON-Monitor 29-43
Using SMI Tables 29-44

Monitoring Logging Activity 29-44
Monitoring Logical-Log Files 29-44
Monitoring the Logical Log for Fullness 29-44
Using Command-Line Utilities 29-45
Using ON-Monitor 29-46
Using SMI Tables 29-47

Monitoring the Physical-Log File 29-47
Using Command-Line Utilities. 29-48
Using ON-Monitor 29-49
Monitoring the Physical-Log and Logical-Log Buffers 29-49
Using Command-Line Utilities. 29-49
Using ON-Monitor 29-51
Using SMI Tables 29-51

Monitoring the Database Server for Disabling I/O Errors 29-52
Using the Message Log to Monitor Disabling I/O Errors 29-52
Using Event Alarms to Monitor Disabling I/O Errors 29-53

Monitoring Disk Usage 29-53
Monitoring Chunks 29-53
Using Command-Line Utilities. 29-54
Using ON-Monitor 29-58
Using SMI Tables 29-59
Monitoring Tblspaces and Extents 29-60
Using Command-Line Utilities. 29-60
Using SMI Tables 29-62
Using System Catalog Tables 29-63
Monitoring TEXT and BYTE Data in a Blobspace 29-63
Using Command-Line Utilities. 29-63
Using ON-Monitor 29-68
Monitoring TEXT and BYTE Data in a Dbspace 29-69
Using Command-Line Utilities. 29-69

Monitoring High-Availability Data-Replication Status 29-71
Using Command-Line Utilities. 29-71
Using ON-Monitor 29-73
Using SMI Tables 29-73

The first part of this chapter describes the types of information that you can monitor and the sources for that information within the database server. It also describes how to monitor the database server. For additional information about monitoring database server performance, see your *Performance Guide*.

Information That You Can Monitor

You can monitor the following types of information:

- Database server configuration information
- Checkpoint information
- Shared-memory information
  - Shared-memory segments
  - Shared-memory profile
  - Buffers
  - Latches
  - Locks
- Active tblspaces
- Virtual processors
- Sessions and threads
- Transactions
- Parallel database query (PDQ) resources and queries
- Databases

- Logging activity
  - Logical-log files
  - Physical-log file
  - Physical-log and logical-log buffers
  - Log backup status
- Disk usage
  - Chunks
  - Tblspaces and extents
  - TEXT or BYTE data in a blobspace
  - TEXT or BYTE data in a dbspace
- High-availability data-replication information

Sources of Information for Monitoring the Database Server

You can use the following sources to gather information about database server activity:

- The message log
- The event alarm
- The system console
- ON-Monitor ♦ (UNIX)
- SMI tables
- The **onstat** and **oncheck** utilities
- The **onperf** utility

Each of these topics is explained in the sections that follow.

What Is the 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 Alarm" on page 29-8. To specify the message-log pathname, set the MSGPATH configuration parameter. For more information, see "MSGPATH" on page 33-55.

Why Read the Message Log?

Informix recommends that you monitor the message log once or twice a day to ensure that processing is proceeding normally. Informix has documented the messages to provide you with as much information as possible about database server processing. The messages are listed in Chapter 36, "Message-Log Messages."

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, Informix does not expect you to do this kind of monitoring.

Changing the Destination for Message-Log Messages

You can change the value of MSGPATH while the database server is in on-line mode, but the changes do not take effect until you reinitialize shared memory.

Monitoring the Message Log

Monitor the message log periodically to verify that the database server operations are 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.

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

Event Alarm

The database server provides a mechanism for automatically triggering administrative actions based on an event that occurs in the database server environment. This mechanism is the event-alarm feature.

To use the event-alarm feature, set the ALARMPROGRAM configuration parameter to the full pathname of an executable file that performs the necessary administrative actions. You must provide this executable file. It can be a shell script or binary program. When any of the events in a predefined set occur, the database server invokes this executable and passes it the following parameters (the executable file must be written to accept these parameters).

Parameter	Data Type
Event severity (see Figure 29-1 for values)	integer
Event class ID (see Figure 29-2 for values)	integer
Event class msg (see Figure 29-2 for values)	string
Event specific msg	string
Event `see also` file	string

Some of the events that the database server reports to the message log cause it to invoke the alarm program. The class messages listed in Figure 29-2 on page 29-10 indicate the events that the database server reports.

For example, if a thread attempts to acquire a lock, but the maximum number of locks specified by LOCKS has already been reached, the database server writes the following message to the message log:

```
10:37:22  Checkpoint Completed:  duration was 0 seconds.
10:51:08  Lock table overflow - user id 30032, rstcb 10132264
10:51:10  Lock table overflow - user id 30032, rstcb 10132264
10:51:12  Checkpoint Completed:  duration was 1 seconds.
```

If you set ALARMPROGRAM to the pathname of an alarm program, the database server passes the following arguments to your alarm program:

```
3
21
Dynamic Server resource overflow: 'Locks'.
Lock table overflow - user id 30032, rstcb 10132264
```

In this example, the database server does not pass a `see also file` value.

Event Severity

The first parameter passed to the alarm program is the event-severity code. All events reported to the message log have one of the severity codes listed in Figure 29-1. Message log events that have severity 1 do not cause the database server to invoke the alarm program.

Figure 29-1
Event-Severity Codes

Severity	Description
1	Not noteworthy. The event is not reported to the alarm program (for example, date change in the message log).
2	Information. No error has occurred, but some routine event completed successfully (for example, checkpoint or log backup completes).
3	Attention. This event does not compromise data or prevent the use of the system; however, it warrants attention (for example, one chunk of a mirrored pair goes down).
4	Emergency. Something unexpected occurred that might compromise data or access to data (assertion failure, or **oncheck** reports data corrupt). Take action immediately.
5	Fatal. Something unexpected occurred and caused the database server to fail.

Event Class ID

An event class ID is an integer that the database server substitutes as the second parameter in your alarm program. Each event class ID is associated with one of the events that causes the database server to run your alarm program. These class IDs are listed in the first column of the table in Figure 29-2.

Class Message

A class message is the text of the message that the database server substitutes for the third parameter of your alarm program when an event causes the database server to run your alarm program. The class messages are listed in Figure 29-2.

Figure 29-2
Class-ID and Class-Message Values

Class ID	Class Message
1	Table failure: '%s' (dbsname:"owner".tabname)
2	Index failure: '%s' (dbsname:"owner".tabname-idxname)
3	Blob failure: '%s' (dbsname:"owner".tabname)
4	Chunk is off-line, mirror is active: %ld (chunk number)
5	DBSpace is off-line: '%s' (dbspace name)
6	Internal Subsystem failure: '%s'
7	Database server initialization failure
8	Physical Restore failed
9	Physical Recovery failed
10	Logical Recovery failed
11	Cannot open Chunk: '%s' (pathname)
12	Cannot open Dbspace: '%s' (dbspace name)
13	Performance Improvement possible

(1 of 2)

Class ID	Class Message
14	Database failure. ‘%s’ (database name)
15	High-availability data-replication failure
16	Archive completed: ‘%s’ (dbspace list)
17	Archive aborted: ‘%s’ (dbspace list)
18	Log Backup completed: %ld (log number)
19	Log Backup aborted: %ld (log number)
20	Logical Logs are full -- Backup is needed
21	Database server resource overflow: ‘%s’ (resource name)
22	Long Transaction detected
23	Logical Log ‘%ld’ (number) Complete
24	Unable to Allocate Memory

(2 of 2)

Specific Messages

The database server substitutes additional information for the fourth parameter of your alarm program. In general, the text of this message is that of the message written to the message log for the event.

See Also Paths

For some events, the database server writes additional information to a file when the event occurs. The pathname in this context refers to the pathname of the file where the database server writes the additional information.

What Is the Console?

The database server sends messages that are useful to the database server administrator by way of the *system console*. To specify the destination pathname of console messages, set the CONSOLE configuration parameter. For more information, see "CONSOLE" on page 33-15.

You can change the value of CONSOLE while the database server is in on-line mode, but the changes do not take effect until you reinitialize shared memory.

UNIX

Monitoring with ON-Monitor

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. See Chapter 32, "ON-Monitor for UNIX."

Monitoring with 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 against them to determine almost anything you might want to know about your database server. For a description of the tables, see "Using the System-Monitoring Interface" on page 34-5.

Monitoring with onstat and oncheck Utilities

The **onstat** and **oncheck** utilities provide a way to monitor database server information from the command line.

You can compare the display options of **oncheck** to **onstat**. 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** describes information that changes dynamically during processing, such as buffers, locks, and users. The **oncheck** utility tends to display mostly configuration and disk-usage information that resides on disk and changes less frequently.

Monitoring with onperf

The database server includes a graphical monitoring tool called **onperf**. This tool can monitor most of the metrics that **onstat** provides. It has 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 on the **onperf** tool, see your *Performance Guide*.

Monitoring with the onstat Banner Line

Whenever the database server is blocked, **onstat** displays the following line after the banner line:

```
Blocked: reason
```

The metavariable *reason* can take one of the following values.

Reason	Description
CKPT	Checkpoint
LONGTX	Long transaction
ARCHIVE	Ongoing archive
MEDIA_FAILURE	Media failure
HANG_SYSTEM	Database server failure
DBS_DROP	Dropping a dbspace
DDR	Discrete high-availability data replication
LBU	Logs full high-water mark

For an example of what **onstat** displays when the database server is blocked to preserve logical-log space for administrative tasks, see "Monitoring the Logical Log for Fullness" on page 29-44.

Monitoring Configuration Information

One of the tasks of the database server administrator is to keep records of the configuration. Methods of obtaining the configuration are described here.

Using Command-Line Utilities

Use the following utilities to monitor configuration information.

onstat -c

Execute **onstat -c** to display a copy of the ONCONFIG file. For information about this file, see Chapter 33, "Configuration Parameters."

Changes to the ONCONFIG file do not take effect until you reinitialize shared memory. If you change a configuration parameter but do not reinitialize shared memory, the effective configuration differs from what the **onstat -c** option displays.

If you do not set the **ONCONFIG** environment variable, the database server displays the contents of the **INFORMIXDIR** file.

oncheck -pr

Execute **oncheck -pr** to obtain the configuration information that the database server stores in the PAGE_CONFIG reserved page. The reserved page contains a description of the current, effective configuration. An example is shown in Figure 29-3. If you change the configuration parameters from the command line and run **oncheck -pr** before you reinitialize shared memory, **oncheck** discovers that values in the configuration file do not match the current values in the reserved pages and returns a warning message.

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

Figure 29-3
oncheck -pr PAGE_CONFIG Output

UNIX

Using ON-Monitor

Select **Status→Configuration**. This option creates a copy of the current, effective configuration and stores 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.

If you modify the configuration parameters but have not yet reinitialized shared memory, the effective parameters might be different than the parameters that are in the ONCONFIG file.

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 *Performance Guide*.

Using Command-Line Utilities

You can use the following command-line utilities to obtain checkpoint information.

onstat -m

Execute **onstat -m** to view the last 20 entries in the message log. If a checkpoint record does not appear in the last 20 entries, read the message log directly with a text editor. The database server writes individual checkpoint records to the log when the checkpoint ends. If a checkpoint check occurs, but the database server has no pages to write to disk, the database server does not write any records to the message log.

onstat -p

Execute **onstat -p** to obtain these checkpoint statistics:

- Number of checkpoints that occurred since the database server was brought on-line (**numckpts**)
- Number of times that a user thread waits for a checkpoint to finish (**ckpwaits**)

  The database server prevents a user thread from entering a *critical section* during a checkpoint.

UNIX

Using ON-Monitor

You can use the following ON-Monitor options to monitor checkpoint information.

Profile Option

Select **Status→Profile**. This option displays the **Checkpoints** and **Check Waits** fields described earlier under **onstat -p**.

Force-Ckpt Option

Select the Force-Ckpt menu. The screen shown in Figure 29-4 is displayed.

```
Do you want to force a checkpoint? (y/n)

Last checkpoint done  :    Fri Jul 29 09:34:33 1995
Last checkpoint check :    Fri Jul 29 13:16:50 1995
```

Figure 29-4
ON-Monitor Force-Ckpt Screen

A checkpoint check occurs if the time specified by the CKPTINTVL configuration parameter has elapsed since the last checkpoint. If no modifications have been made since the time of the last checkpoint, the database server does not perform a checkpoint; that is, the database server does not flush the physical-log buffers to disk. The time in the **Last Checkpoint Done** field does not change until a checkpoint occurs.

Using SMI Tables

The **sysprofile** table provides the same checkpoint statistics that are available from the following sources:

- **onstat -p** command
- Profile option in ON-Monitor ♦

UNIX

The **sysprofile** table contains two columns, **name** and **value**. The **name** column contains the statistic name, and the **value** column contains the statistic value. These rows contain the following checkpoint information:

numckpts is the number of checkpoints that have occurred since the database server was brought on-line.

ckptwts is the number of times that threads waited for a checkpoint to finish to enter a *critical section* during a checkpoint.

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 using 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 "SHMVIRTSIZE" on page 33-83.

The **onstat -g seg** command lists information for each shared-memory segment, including the address and size of the segment. Example output is shown in Figure 29-5.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:45:34 -- 4600 Kbytes

Segment Summary:
 (resident segments are not locked)
id       key        addr     size       ovhd     class blkused  blkfree
300      1381386241 400000   614400     800      R     71       4
301      1381386242 496000   4096000    644      V     322      178
```

Figure 29-5
onstat -g seg Output

Monitoring Shared-Memory Profile

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.

Using Command-Line Utilities

Execute **onstat -p** to display statistics on database server activity. These statistics are shown in Figure 29-6.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:41:04 -- 8920 Kbytes

Profile
dskreads pagreads bufreads %cached dskwrits pagwrits bufwrits %cached
382      400      14438    97.35   381      568      3509     89.14

isamtot  open     start    read     write    rewrite  delete   commit   rollbk
9463     1078     1584     2316     909      162      27       183      1

ovlock   ovuserthread ovbuff   usercpu  syscpu   numckpts flushes
0        0            0        13.55    13.02    5        18

bufwaits lokwaits lockreqs deadlks  dltouts  ckpwaits compress seqscans
14       0        16143    0        0        0        101      68

ixda-RA  idx-RA   da-RA    RA-pgsused lchwaits
5        0        204      148        12
```

Figure 29-6
onstat -p Output

The **onstat -p** output contains several fields that are not included in the information that the ON-Monitor Profile option displays. For a description of all the fields displayed by this option, see "-p Option" on page 35-84.

UNIX

Using ON-Monitor

Select **Status→Profile**. The screen displays shared-memory statistics, as well as the current operating mode, the boot time, and the current time.

The field labels on the ON-Monitor Profile screen are easier to understand and are arranged in a slightly different order than the fields that appear if you execute **onstat -p**. However, all these statistics are included in the **onstat -p** output.

Using SMI Tables

Query the **sysprofile** table to obtain shared-memory statistics. This table contains all of the statistics available in **onstat -p** 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 *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 -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**)

Figure 29-7 shows these fields.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:41:04 -- 8920 Kbytes

Profile
dskreads pagreads bufreads %cached dskwrits pagwrits bufwrits %cached
382      400      14438    97.35   381      568      3509     89.14
:
```

Figure 29-7
Cached Read and Write Statistics in the onstat -p Output

The number of reads or writes can appear as a negative number if the number of occurrences exceeds 2^{32}.

The **onstat -p** option also displays a statistic (**bufwaits**) that indicates the number of times that sessions had to wait for a buffer.

onstat -B

Execute **onstat -B** to obtain the following buffer information:

- Address of every regular shared-memory buffer
- Page numbers for all pages that remain in shared memory
- Address of the thread that currently holds the buffer
- Address of the first thread that is waiting for each buffer

An example of **onstat -B** output is shown in Figure 29-8.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:21:46 -- 8920 Kbytes

Buffers
address  userthread flgs pagenum  memaddr  nslots pgflgs xflgs owner    waitlist
849ae8   0          86   100955   84e000   1      b0     0     0        0
849b40   0          6    10095b   84e800   0      4      0     0        0
849b98   0          6    1009eb   84f000   0      4      0     0        0
849bf0   0          6    1008f5   84f800   2      70     0     0        0
:
84dea0   0          86   10093e   8b0800   8      1      0     0        0
84def8   0          6    10094b   8b1000   0      4      0     0        0
84df50   0          86   1009cd   8b1800   9      b0     0     0        0
 0 modified, 200 total, 256 hash buckets, 2048 buffer size
```

Figure 29-8
onstat -B Output

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

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. Example output is shown in Figure 29-9. For more information on the fields displayed by this option, see "-b Option" on page 35-70.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:12:23 -- 8920 Kbytes

Buffers
address  userthread flgs pagenum  memaddr  nslots pgflgs xflgs owner    waitlist
84a748   0          27   1012b0   860000   19     2001   80    8067c4   0
84add0   0          0    101752   869800   19     2001   80    807890   0
84b2a0   0          27   100c31   870800   19     2001   80    8067c4   0
84c798   0          27   10108e   88f000   19     2001   80    8067c4   0
84d818   0          27   101272   8a7000   19     2001   80    8067c4   0
 154 modified, 200 total, 256 hash buckets, 2048 buffer size
```

Figure 29-9
onstat -b Output

onstat -X

Execute **onstat -X** to obtain the same information as **onstat -b**, along with the *complete* list of all threads that are waiting for buffers, not just the first waiting thread.

UNIX

Using ON-Monitor

To access the fields mentioned on page 29-20 for **onstat -p** (**bufreads**, **%cached**, **bufwrits**, **%cached**), select **Status→Profile**. The output is shown in Figure 29-10.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:33:33 -- 8456 Kbytes
:
Disk Reads  Buff. Reads    %Cached  Disk Writes  Buff. Writes    %Cached
       177          330      46.36            4             0       0.00
:
```

Figure 29-10
Cached Read and Write Statistics in the Profile Option of the ON-Monitor Status Menu

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 *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 following command-line utilities to obtain information on buffer-pool activity.

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 the BUFFERS parameter in the ONCONFIG file. Figure 29-11 shows **onstat -p** output, including the **ovbuff** field.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:41:04 -- 8920 Kbytes
.
.
.
ovtbls   ovlock   ovuserthread ovbuff   usercpu  syscpu   numckpts flushes
0        0        0            0        13.55    13.02    5        18
.
.
.
```

Figure 29-11
onstat -p Output Showing ovbuff Field

onstat -F

Execute **onstat -F** to obtain a count of the writes performed by write type. (For an explanation of the different write types, see "How Write Types Describe Flushing Activity" on page 11-51.) An example of the output is shown in Figure 29-12 on page 29-25. This information tells you when and how the buffers are flushed.

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

An example of the **onstat -F** output is shown in Figure 29-12. For more information on the **onstat -F** fields, see "-F Option" on page 35-75.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:43:35 -- 8920 Kbytes

Fg Writes     LRU Writes    Chunk Writes
0             146           140

address  flusher  state    data
8067c4   0        I        0        = 0X0
      states: Exit Idle Chunk Lru
```

Figure 29-12
onstat -F Output

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 more information on this option, see "-R Option" on page 35-87.) Figure 29-13 shows an example of **onstat -R** output.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:29:42 -- 4584 Kbytes

8 buffer LRU queue pairs
# f/m   length   % of   pair total
 0 f          3    37.5%       8
 1 m          5    55.6%
 2 f          5    45.5%      11
 3 m          6    54.5%
 4 f          2    18.2%      11
 5 m          9    81.8%
 6 f          5    50.0%      10
 7 m          5    55.6%
 8 F          5    50.0%      10
 9 m          5    45.5%
10 f          0     0.0%      10
11 m         10   100.0%
12 f          1    11.1%       9
13 m          8    88.9%
14 f          2    28.6%       7
15 m          5    71.4%
53 dirty, 76 queued, 80 total, 128 hash buckets, 2048 buffer size
start clean at 60% (of pair total) dirty, or 6 buffs dirty, stop at 50%
```

Figure 29-13
onstat -R Output

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

Monitoring Latches

You can obtain statistics on latch use and information on specific latches.

The statistics include the number of requests for latches and the number of times that threads had to wait to obtain a latch. These statistics give you a measure of the system activity.

Information on specific latches includes a listing of all the latches that are held by a thread and any threads that are waiting for latches. This information allows you to locate any specific resource contentions that exist.

Using Command-Line Utilities

You can use the following command-line utilities to obtain information about latches.

onstat -p

Execute **onstat -p** to obtain the values in the fields **lchreqs** and **lchwaits**. These fields store the number of requests for a latch and the number of times that a thread was required to wait for a shared-memory latch. A large number of latch waits typically results from a high volume of processing activity in which the database server is logging most of the transactions. (The administrator cannot configure or tune the number of latches; the database server sets this function internally.) Figure 29-14 shows **onstat -p** output, including the **lchreqs** and **lchwaits** fields.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:41:04 -- 8920 Kbytes
.
.
.
ixda-RA  idx-RA   da-RA    RA-pgsused lchreqs  lchwaits
5        0        204      148        151762   12
```

Figure 29-14
onstat -p Output Showing lchwaits Field

onstat -s

Execute **onstat -s** to obtain general latch information. The output includes the **userthread** column, which lists the address of any user thread that is waiting for a latch. (See Figure 29-15.) You can compare this address with the user addresses in the **onstat -u** output to obtain the user-process identification number.

Warning: *Never kill a database server process that is holding a latch. If you do, the database server immediately initiates an abort.*

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:13:47 -- 4664 Kbytes

Latches with lock or userthread set
name     address  lock wait userthread
LRU1     402e90   0    0        6b29d8
bf[34]   4467c0   0    0        6b29d8
```

Figure 29-15
onstat -s Output

UNIX

Using ON-Monitor

To access the **latch Waits** field, mentioned on page 29-27 for **onstat -p**, select **Status→Profile**.

Using SMI Tables

Query the **sysprofile** table to obtain the number of requests for a latch and the number of times a thread had to wait for a latch. The following rows are relevant.

Row	Description
latchreqs	Number of requests for a latch
latchwts	Number of times that a thread had to wait for a latch

Monitoring Locks

You can obtain profile statistics on lock use and information on specific locks.

The statistics include the number of times that threads attempted to exceed the maximum number of locks, the number of times that threads had to wait for a lock, and the number of times that threads requested a lock. This information indicates whether the number of locks is appropriate and provides a measure of the database server concurrency.

Information on specific locks includes a listing of the locks that are held by a thread. This information allows you to locate a source of contention.

Using Command-Line Utilities

onstat -p

The **onstat -p** option displays the following three lock statistics:

- The number of times that sessions attempted to exceed the maximum number of locks specified by the LOCKS parameter (**ovlock**)
- The number of times that sessions had to wait for a lock (**lokwaits**)
- The number of times that sessions requested a lock (**lockreqs**)

Figure 29-16 shows lock statistics.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:27:06 -- 9430 Kbytes
.
.
.
ovtbls   ovlock   ovuserthread ovbuff   usercpu  syscpu   numckpts flushes
0        0        0            0        13.55    13.02    5        18

bufwaits lokwaits lockreqs deadlks  dltouts  ckpwaits compress seqscans
14       0        16143    0        0        0        101      68
.
.
.
```

Figure 29-16
onstat -p Output Showing Lock Statistics

onstat -k

The **onstat -k** option displays information about active locks. The following information is displayed:

- The user session that owns the lock (**owner**)
- The type of the lock (**type**)
- The scope of the lock (**rowid**)

You can determine the type of the lock from the flags in the type column. For example, a shared lock has an S flag displayed. You can determine the scope of the lock from the value in the rowid column. For example, a zero in this column always indicates a table lock. (For more information on the fields displayed, see page 35-79.) An example of the output displayed by this option is shown in Figure 29-17.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:27:37 -- 8920 Kbytes

Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
8109e0   0        809d84   0        HDR+S    100002   203        0
810a08   0        809a28   0            S    100002   203        0
810a30   0        8096cc   0            S    100002   203        0
  .
  .
  .
810c10   0        8096cc   810a30       IX   10006a   0          0
810c38   0        808cb8   810aa8   HDR+IX   10006a   0          0
810c88   0        809014   810bc0   HDR+U    10006a   33e04      0
 15 active, 2000 total, 128 hash buckets
```

Figure 29-17
onstat -k Output

UNIX

Using ON-Monitor

To monitor the same three lock statistics that are mentioned on page 29-29 for the **onstat -p** option, select **Status→Profile**.

Using SMI Tables

Query the **sysprofile** table to obtain statistics on lock use. The following rows contain the relevant statistics.

Row	Description
ovlock	Number of times that sessions attempted to exceed the maximum number of locks
lockreqs	Number of times that sessions requested a lock
lockwts	Number of times that sessions had to wait for a lock

Query the **syslocks** table to obtain information on each active lock. The **syslocks** table contains the following columns.

Column	Description
dbsname	Database on which the lock is held
tabname	Name of the table on which the lock is held
rowidlk	ID of the row on which the lock is held (0 means table lock)
keynum	The keynum for the row
type	Type of lock
owner	Session ID of the lock owner
waiter	Session ID of the first waiter on the lock

Monitoring Active Tblspaces

Monitor tblspaces to determine which tables are active. Active tables are those that are currently open to a thread.

Using Command-Line Utilities

The **onstat -t** output includes the tblspace number and the following four fields.

Field	Description
npages	Pages allocated to the tblspace
nused	Pages used from this allocated pool
nextns	Number of extents used
npdata	Number of data pages used

If a specific operation needs more pages than are available (**npages** minus **nused**), a new extent is required. If enough space is available in this chunk, the database server allocates the extent here; if not, the database server looks for space in other available chunks. If none of the chunks contains adequate contiguous space, the database server uses the largest block of contiguous space that it can find in the dbspace. An example of the output from this option is shown in Figure 29-18.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:20:00 -- 4584 Kbytes

Tblspaces
 n address  flgs ucnt tblnum   physaddr npages nused  npdata nrows  nextns
 0 422528   1    1    100001   10000e   150    124    0      0      3
 1 422640   1    1    200001   200004   50     36     0      0      1
54 426038   1    6    100035   1008ac   3650   3631   3158   60000  3
62 4268f8   1    6    100034   1008ab   8      6      4      60     1
63 426a10   3    6    100036   1008ad   368    365    19     612    3
64 426b28   1    6    100033   1008aa   8      3      1      6      1
193 42f840  1    6    10001b   100028   8      5      2      30     1
 7 active, 200 total, 64 hash buckets
```

Figure 29-18
onstat -t Output

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.

Using Command-Line Utilities

You can use the following command-line utilities to monitor virtual processors.

onstat -g glo

This command displays information about each virtual processor that is currently running, as well as cumulative statistics for each virtual processor class. An example of the output from this option is shown in Figure 29-19.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 02:27:42 -- 4664 Kbytes

MT global info:
sessions threads  vps      lngspins
1        15       8        0

Virtual processor summary:
class    vps      usercpu          syscpu           total
 cpu       3       479.77          190.42          670.18
 aio       1         0.83            0.23            1.07
 pio       1         0.42            0.10            0.52
 lio       1         0.27            0.22            0.48
 soc       0         0.00            0.00            0.00
 tli       0         0.00            0.00            0.00
 shm       0         0.00            0.00            0.00
 adm       1         0.10            0.45            0.55
 opt       0         0.00            0.00            0.00
 msc       1         0.28            0.52            0.80
 adt       0         0.00            0.00            0.00
 total     8       481.67          191.93          673.60

Individual virtual processors:
   vp        pid         class    usercpu          syscpu           total
    1        1776          cpu     165.18           40.50          205.68
    2        1777          adm       0.10            0.45            0.55
    3        1778          cpu     157.83           98.68          256.52
    4        1779          cpu     156.75           51.23          207.98
    5        1780          lio       0.27            0.22            0.48
    6        1781          pio       0.42            0.10            0.52
    7        1782          aio       0.83            0.23            1.07
    8        1783          msc       0.28            0.52            0.80
                           tot     481.67          191.93          673.60
```

Figure 29-19
onstat -g glo Output

onstat -g ioq

Use the **onstat -g ioq** option to determine whether you need to allocate additional AIO virtual processors. The command **onstat -g ioq** displays the length of the I/O queues under the column **len**, as shown in Figure 29-20.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:57:59 -- 4584 Kbytes

AIO I/O queues:
class/hvp-id len maxlen totalops  dskread dskwrite  dskcopy
  kio   0      0     68    18940    14081     4859        0
  kio   1      0     62    18906    13167     5739        0
  kio   2      0     50    18632    13659     4973        0
  msc   0      0      1       55        0        0        0
  aio   0      0      1       45       45        0        0
  pio   0      0      0        0        0        0        0
  lio   0      0      0        0        0        0        0
```

Figure 29-20
onstat -g ioq Output

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.

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 have to add more of those virtual processors to your configuration. Figure 29-21 displays **onstat -g rea** output.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:07:39 -- 4584 Kbytes

Ready threads:
tid     tcb      rstcb   prty    status                vp-class   name

6       536a38  406464  4       ready                   3cpu    main_loop()
28      60cfe8  40a124  4       ready                   1cpu    onmode_mon
33      672a20  409dc4  2       ready                   3cpu    sqlexec
```

Figure 29-21
onstat -g rea Output

Using 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

Monitoring Sessions and Threads

Monitor sessions and threads to determine how many threads are active and the shared-memory resources that those threads are using. This information allows you to determine if an application is using a disproportionate amount of the resources.

For information on how to monitor queries and PDQ resources, see your *Performance Guide*.

Using Command-Line Utilities

You can use the following command-line utilities to monitor sessions and threads.

onstat -u

The **onstat -u** utility displays information on all active threads that require an RSAM task control block (rstcb) structure. Active threads include threads that belong to user sessions, as well as some that correspond to server daemons (for example, page cleaners). An example of output from this utility is shown in Figure 29-22 on page 29-36.

The utility displays a table that contains the following information:

- The address of each thread
- Flags that indicate the present state of the thread (for example, waiting on a buffer, waiting for a checkpoint), whether the thread is the primary thread for a session, and what type of thread it is (for example, user thread, daemon thread and so on). For information on these flags, see "-u Option" on page 35-90.
- The sessid and user login ID for the session to which the thread belongs. A sessid of 0 indicates a daemon thread.
- Whether the thread is waiting for a specific resource and the address of that resource
- The number of locks that the thread is holding
- The number of read calls and the number of write calls that the thread has executed
- The maximum number of concurrent user threads that were allocated since you last initialized the database server

If you execute **onstat -u** while the database server is performing fast recovery, several server threads might appear in the display.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:50:22 -- 8896 Kbytes

Userthreads
address  flags   sessid  user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0       informix -        0        0    0     33       19
80ef18   ---P--F 0       informix -        0        0    0     0        0
80f2a4   ---P--B 3       informix -        0        0    0     0        0
80f630   ---P--D 0       informix -        0        0    0     0        0
80fd48   ---P--- 45      chrisw   ttyp3    0        0    1     573      237
810460   ------- 10      chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42      chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10      chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47      chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46      chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10      chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43      chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10      chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10      chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44      chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

Figure 29-22
onstat -u Output

onstat -g ath

Use the **onstat -g ath** option to obtain a listing of all threads. Unlike the **onstat -u** option, this listing includes internal daemon threads that do not have an RSAM control block. On the other hand, the **onstat -g ath** display does not include the sessid (because not all threads belong to sessions).

Threads that have been started by a primary decision-support thread have a name that indicates their role in the decision-support query. For example, in Figure 29-23, four scan threads that belong to a decision-support thread are displayed.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:50:15 -- 8896 Kbytes

Threads:
tid     tcb     rstcb   prty    status                  vp-class   name
:
11      994060  0       4       sleeping(Forever)         1cpu     kaio
12      994394  80f2a4  2       sleeping(secs: 51)        1cpu     btclean
26      99b11c  80f630  4       ready                     1cpu     onmode_mon
32      a9a294  812b64  2       ready                     1cpu     sqlexec
113     b72a7c  810b78  2       ready                     1cpu     sqlexec
114     b86c8c  81244c  2       cond wait(netnorm)        1cpu     sqlexec
115     b98a7c  812ef0  2       cond wait(netnorm)        1cpu     sqlexec
116     bb4a24  80fd48  2       cond wait(netnorm)        1cpu     sqlexec
117     bc6a24  81161c  2       cond wait(netnorm)        1cpu     sqlexec
118     bd8a24  811290  2       ready                     1cpu     sqlexec
119     beae88  810f04  2       cond wait(await_MC1)      1cpu     scan_1.0
120     a8ab48  8127d8  2       ready                     1cpu     scan_2.0
121     a96850  810460  2       ready                     1cpu     scan_2.1
122     ab6f30  8119a8  2       running                   1cpu     scan_2.2
```

Figure 29-23
onstat -g ath Output

onstat -g act

Use the **onstat -g act** option to obtain a list of active threads.

onstat -g ses

Use the **onstat -g ses** option to monitor the resources allocated for, and used by, a session—in particular, a session that is running a decision-support query. For example, in Figure 29-24, session number 49 is running five threads for a decision-support query.

Figure 29-24
onstat -g ses Output

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:04:36 -- 8896 Kbytes

session                                      #RSAM    total      used
id       user     tty      pid   hostname  threads  memory     memory
57       informix -        0     -         0        8192       5908
56       user_3   ttyp3    2318  host_10   1        65536      62404
55       user_3   ttyp3    2316  host_10   1        65536      62416
54       user_3   ttyp3    2320  host_10   1        65536      62416
53       user_3   ttyp3    2317  host_10   1        65536      62416
52       user_3   ttyp3    2319  host_10   1        65536      62416
51       user_3   ttyp3    2321  host_10   1        65536      62416
49       user_1   ttyp2    2308  host_10   5        188416     178936
2        informix -        0     -         0        8192       6780
1        informix -        0     -         0        8192       4796
```

onstat -g sts

Use the **onstat -g sts** option to obtain information on stack-size use for each thread. The output includes the following fields:

- The thread ID
- The maximum stack size configured for each thread
- The maximum stack size used by the thread

You can use the output of the threads that belong to user sessions to determine if you need to alter the maximum stack size configured for a user session. To alter the maximum stack size for all user sessions, change the value of the STACKSIZE configuration parameter. To alter the maximum stack size for a single user session, change the value of the **INFORMIXSTACKSIZE** environment variable. For more information, see "STAGEBLOB" on page 33-86 and the description of **INFORMIXSTACKSIZE** in the *Informix Guide to SQL: Reference*.

UNIX

Using ON-Monitor

Select **Status→User**. The display (shown in Figure 29-25) provides a subset of the information displayed by the **onstat -u** utility. The following information is displayed:

- The session ID
- The user ID
- The number of locks that the thread is holding
- The number of read calls and write calls that the thread has executed
- Flags that indicate the present state of the thread (for example, waiting on a buffer, waiting for a checkpoint), whether the thread is the primary thread for a session, and what type of thread it is (for example, user thread, daemon thread, and so on)

```
                        USER THREAD INFORMATION

                    Locks       Disk       Disk       User thread
Session   User      Held        Reads      Writes     Status

      0   informix     0          96          2       ------D
      0   informix     0           0          0       ------F
      0   informix     0           0          0       -------
     15   informix     0           0          0       Y-----M
      0   informix     0           0          0       ------D
     17   chrisw       1           3         34       Y------
```

Figure 29-25
Output from the User Option of the ON-Monitor Status Menu

Using SMI Tables

Query the **syssessions** table to obtain the information in the following columns.

Column	Description
sid	Session ID
username	User name (login ID) of the user
uid	User ID

(1 of 2)

Column	Description
pid	Process ID
connected	Time that the session started
feprogram	Application that is running as the client (front-end program)

(2 of 2)

In addition, some columns contain flags that indicate if the *primary* thread of the session is waiting for a latch, lock, log buffer, or transaction; if it is an ON-Monitor thread; and if it is in a critical section. (ON-Monitor is available only on UNIX.) For a full list of the **syssessions** columns, see "syssessions" on page 34-26.

Important: *The information in the* ***syssessions*** *table is organized by session, while the information displayed by* ***onstat -u*** *is organized by thread. Also, unlike the* ***onstat -u*** *option, the* ***syssessions*** *table does not include information on daemon threads, only user threads.*

Query the **syssesprof** table to obtain a profile of the activity of a session. This table contains a row for each session with columns that store statistics on session activity (for example, number of locks held, number of row writes, number of commits, number of deletes, and so on). For the full description of these columns, see "syssesprof" on page 34-24.

Monitoring PDQ Resources and Queries

For parallel database queries (PDQ), you can monitor:

- resources that the resource grant manager (RGM) allocates for PDQ query and the resources that those queries currently use.
- threads used to execute a PDQ query.

For more information on the RGM, PDQ, and how to monitor a PDQ query, see your *Performance Guide*.

Monitoring Transactions

Monitor transactions to track open transactions and the locks held by those transactions.

Using Command-Line Utilities

You can use the following command-line options to monitor transactions.

onstat -x

The **onstat -x** output contains the following information for each open transaction:

- The address of the transaction in shared memory
- Flags that indicate the following information:
  - The present state of the transaction (thread attached, suspended, waiting for a rollback)
  - What stage the transaction is in (BEGIN WORK, prepared to commit, committing / committed, rolling back)
  - The nature of the transaction (global transaction, coordinator, subordinate, both coordinator and subordinate)
- The thread that owns the transaction
- The number of locks held by the transaction
- The logical-log file in which the BEGIN WORK record was logged
- The isolation level
- The number of attempts to start a recovery thread
- The coordinator for the transaction (if the transaction is being executed by a subordinate)
- The maximum number of concurrent transactions since you last initialized the database server

This utility is especially useful for monitoring global transactions. For example, you can determine whether a transaction has been heuristically rolled back. Example output is shown in Figure 29-26.

Figure 29-26
onstat -x Output

```
Transactions
address  flags userthread locks log begin isolation retrys coordinator
40a7e4   A---- 406464     0     0         COMMIT    0
40a938   A---- 4067c4     0     0         COMMIT    0
40aa8c   A---- 406b24     0     0         COMMIT    0
40abe0   A---- 40a124     0     0         COMMIT    0
40ad34   A---- 4093a4     1     0         NOTRANS   0
40ae88   A---- 40a484     2     0         NOTRANS   0
40afdc   A---- 409a64     2     0         NOTRANS   0
40b130   A---- 409704     2     0         NOTRANS   0
40b284   A---- 409dc4     1     0         NOTRANS   0
40b3d8   A---- 409044     3     0         NOTRANS   0
40b52c   A---- 408ce4     2     0         NOTRANS   0
 11 active, 20 total, 6 maximum concurrent
```

onstat -g sql session-id

To obtain summary information about the last SQL statement executed by each session, issue the **onstat -g sql** command with the appropriate session-id. An example of the output for this option is shown in Figure 29-27. For more information on this option, see "-g Monitoring Options" on page 35-76.

Figure 29-27
onstat -g sql Output

```
RSAM Version 7.30.UC1 -- On-Line -- Up 07:00:24 -- 8920 Kbytes

Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
17    ALTER TABLE    mydemo             CR  Not Wait   0    0    7.30.UC10
```

Monitoring Databases

Monitor the databases managed by the database server to determine the logging status of those databases.

UNIX

Using ON-Monitor

To find out the logging status of a database from within ON-Monitor, select the Status menu, Databases option. Figure 29-28 shows the output of this option.

```
                                                    When       Log
Database Name        Owner       In Dbspace         Created    Status

sysmaster            informix    rootdbs            07/28/95     U
mydemo               chrisw      rootdbs            09:08:10     U
```

Figure 29-28 *Output from the Databases Option of the ON-Monitor Status Menu*

When the databases screen appears, ON-Monitor displays the current logging status of each database. 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 next section. The database server uses the following characters to represent database-logging status.

Character	Description
N	No logging
B	Buffered logging
U	Unbuffered logging
A	ANSI compliant

Using SMI Tables

Query the **sysdatabases** table to determine the logging status. This table contains a row for each database managed by the database server. Columns contain flags that indicate the logging status of the database. For a description of the columns in this table, see "sysdatabases" on page 34-14.

Monitoring Logging Activity

This section discusses how to monitor the logical-log files, the physical-log file, logical-log buffers, and physical-log buffers.

Monitoring Logical-Log Files

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). This information is important for logical-log management.

Monitoring the Logical Log for Fullness

When you set LBU_PRESERVE to 1, and the database server is blocking to preserve log space for administrative tasks, the **onstat** utility displays the following message just after its banner line:

```
Blocked: LBU
```

For example, suppose that the database server is running under the following conditions:

- You set LBU_PRESERVE to 1.
- Every log except the last one is full.

In these circumstances, the first two lines of any of the **onstat** options appear as shown in the following example:

```
RSAM Version 7.30.UC1 -- Informix Dynamic Server -- Up 00:12:53
-- 5152 Kbytes
Blocked: LBU
```

Using Command-Line Utilities

You can use the following command-line utilities to monitor logical-log files.

onstat -l

The **onstat -l** utility display consists of the following three sections: physical-log information, logical-log information (general), and information on the individual logical-log files.

The third section contains the following information for each logical-log file:

- The address of the logical-log file descriptor
- The logical-log file logid number
- Status flags that indicate the status of each log

  Flags indicate whether the log is 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

For more information, see "-l Option" on page 35-80. Figure 29-29 shows example output.

```
.
.
address  number   flags    uniqid   begin        size     used    %used
846640   1        F------  0        100233        250        0     0.00
84665c   2        F------  0        10032d        250        0     0.00
846678   3        U---C-L  3        100427        250      175    70.00
846694   4        F------  0        100521        250        0     0.00
8466b0   5        F------  0        10061b        250        0     0.00
```

Figure 29-29
onstat -l Output Showing Logical-Log File Status

oncheck -pr

Execute **oncheck -pr** to obtain logical-log file information stored in the reserved pages dedicated to checkpoint information (PAGE_1CKPT and PAGE_2CKPT). 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. An example of the output is shown in Figure 29-30. For more information, see "Using -pr Option to Display Reserved-Page Information" on page 35-15.

```
:
:
Log file number                 1
Log file flags                  0
Time stamp                      6964
Date/Time file filled           07/28/95 14:48:32
Unique identifier               0
Physical location               100233
Log size                        250
Number pages used               0
:
:
```

Figure 29-30 *oncheck -pr Output Containing Logical-Log File Information*

UNIX

Using ON-Monitor

The Logs option of the Status menu 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. An example of the output is shown in Figure 29-31.

```
:
:
INDIVIDUAL LOG FILES:
Number   Flags      Uniqid   Dbspace                 Pages       Used   % Used
   1    F------          0   rootdbs                   250          0     0.00
   2    F------          0   rootdbs                   250          0     0.00
   3    U---C-L          3   rootdbs                   250        175    70.00
   4    F------          0   rootdbs                   250          0     0.00
```

Figure 29-31 *Output from the Logs Option of the ON-Monitor Status Menu*

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 dbspace 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

Monitoring the Physical-Log File

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.

Using Command-Line Utilities

You can use the following command-line utilities to obtain information about the physical-log file.

onstat -l

The first part of the **onstat -l** display contains the following information:

- The page number of the first page in the physical-log file
- The size of the physical-log file
- The current position in the log where the next write occurs
- The number of pages in the log that have been used
- The percentage of the total physical-log pages that have been used

An example of the **onstat -l** output that contains the physical-log information is shown in Figure 29-32.

```
RSAM Version 7.30.UC1-- On-Line -- Up 08:16:00 -- 8920 Kbytes

Physical Logging
Buffer bufused  bufsize  numpages numwrits pages/io
  P-2  0        16       110      10       11.00
       phybegin physize  phypos   phyused  %used
       10003f   500      233      0        0.00
.
.
.
```

Figure 29-32
onstat -l Output Showing Physical-Log Information

oncheck -pr

Execute **oncheck -pr** to obtain the physical-log file information that the database server stores in those reserved pages dedicated to checkpoint information (PAGE_1CKPT and PAGE_2CKPT). This information gives you the state of the physical log at the last checkpoint. An example of the relevant output is shown in Figure 29-33.

```
Validating Informix Dynamic Server reserved pages - PAGE_1CKPT & PAGE_2CKPT
          Using check point page PAGE_2CKPT.

    Time stamp of checkpoint       16024
    Time of checkpoint             07/30/95 09:34:33
    Physical log begin address     10003f
    Physical log size              500
    Physical log position at Ckpt  e9
.
.
.
```

Figure 29-33
oncheck -pr Output That Includes Physical-Log Information

UNIX

Using ON-Monitor

All the information on the physical-log file that the **onstat -l** utility provides is also available when you select **Status→Logs**.

Monitoring the Physical-Log and Logical-Log Buffers

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 *Performance Guide*.

Using Command-Line Utilities

The **onstat -l** option displays the following information for each physical-log buffer:

- The number of buffer pages used
- The size of each physical log buffer in pages
- The number of pages written to the buffer

- The number of writes from the buffer to disk
- The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)

The following information is available for each logical-log buffer:

- The number of buffer pages used
- The size of each logical-log buffer in pages
- The number of records written to the buffer
- The number of pages written to the buffer
- The number of writes from the buffer to disk
- The ratio of records to pages in the buffer (this is a function of the type of operation)
- The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)

Example output from the **onstat -l** option that contains the relevant fields is shown in Figure 29-34.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 08:16:00 -- 8920 Kbytes

Physical Logging
Buffer bufused  bufsize  numpages numwrits pages/io
  P-2  0        16       110      10       11.00
       phybegin physize  phypos   phyused  %used
       10003f   500      233      0        0.00

Logical Logging
Buffer bufused  bufsize  numrecs  numpages numwrits recs/pages pages/io
  L-1  0        16       3075     162      75       19.0       2.2
.
.
.
```

Figure 29-34
onstat -l Output Showing Log-Buffer Information

UNIX

Using ON-Monitor

All the information that the **onstat -l** utility provides on the physical-log and logical-log buffers is also available if you select **Status→Logs**. An example of the output is shown in Figure 29-35.

```
PHYSICAL LOG:
Buffer  Bufsize     Bufused   Numpages  Numwrites   pages/IO
 P-1         16           0          0          0   0.00
       Phybegin     Physize     Phypos    Phyused   % Used
         10003f         500        236          0   0.00


LOGICAL LOG:
Buffer  Bufsize   Bufused    Numrecs   Numpages  Numwrites  Recs/Page  Pages/IO
 L-2         16         0          1          1          1     1.00       1.00
```

Figure 29-35
Output from the Logs Option of the ON-Monitor Status Menu

Using SMI Tables

Query the **sysprofile** table to obtain statistics on the physical-log and logical-log buffers. The following rows contain the relevant 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

Monitoring the Database Server for Disabling I/O Errors

The database server notifies you about disabling I/O errors in two ways: the message log and 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 Dynamic 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 ONDBSPDOWN, as described in the following table.

ONDBSPDOWN Setting	Result	Action
CONTINUE	Dbspace/blobspace {*space-name*} is disabled.	Restore dbspace/blobspace {*space-name*}.
ABORT	The database server must abort.	Reinitialize shared memory.
WAIT	The database server blocks at next checkpoint.	Use **onmode -k** to shut down, or use **onmode -O** to override**.**

For more information on how to interpret messages that the database server sends to the message log, see Chapter 36, "Message-Log Messages."

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 and make the database server aware of the location of the executable file that you write, see "Sources of Information for Monitoring the Database Server" on page 29-6.

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 Chapter 38, "Disk Structures and Storage."

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.

Using Command-Line Utilities

You can use the following command-line utilities to obtain information about chunks.

Using onstat -d

The **onstat -d** utility lists all dbspaces and blobspaces 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 approximate number of free blobpages
- The pathname of the physical device

The dbspace flags indicate whether a dbspace (or blobspace) is mirrored. The chunk flags provide the following information:

- Whether the chunk is the primary chunk or the mirrored chunk
- Whether the chunk is on-line, is down, is being recovered, or is a new chunk that requires a level-0 dbspace backup before mirroring can become active

Example output for **onstat -D**, which displays the same information plus two additional fields, is shown in Figure 29-36 on page 29-55. For descriptions of the **onstat -d** flags, see "-d Option" on page 35-72.

onstat -D

The **onstat -D** option displays the same information as **onstat -d**, plus the following two fields:

- The number of pages read from the chunk (**page Rd**)
- The number of pages written to the chunk (**page Wr**)

Example output is shown in Figure 29-36.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 00:01:03 -- 4584 Kbytes

Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
40d100   1        1        1        1        N        informix rootdbs
40d144   2        2        2        1        M        informix cookedspace
40d188   3        10       3        1        N B      informix cookedblob
 3 active, 10 total

Chunks
address  chk/dbs offset   page Rd  page Wr  pathname
40c274   1   1   0        146      4        /home/server/root_chunk
40c30c   2   2   0        1        0        /home/server/test_chunk
40c8fc   2   2   0        36       0        /home/server/test_mirr
40c3a4   3   3   0        4        0        /home/server/blob_chunk
 3 active, 10 total
```

Figure 29-36
onstat -D Output

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. Example output is shown in Figure 29-37 on page 29-56.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 01:36:56 -- 8856 Kbytes

AIO global files:
gfd pathname         totalops  dskread dskwrite  io/s
  3 raw_chunk           38808    27241    11567   6.7
  4 cooked_chk1          7925     5660     2265   1.4
  5 cooked_chk2          3729     2622     1107   0.6
```

Figure 29-37
onstat -g iof Output

oncheck -pr

Execute **oncheck -pr** to obtain the chunk information that the database server stores in the reserved pages PAGE_1PCHUNK and PAGE_2PCHUNK. 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. Example output is shown in Figure 29-38.

```
Validating Informix Dynamic Server reserved pages - PAGE_1DBSP & PAGE_2DBSP
          Using dbspace page PAGE_2DBSP.

    DBSpace number                1
    Flags                         1          No mirror chunks
    First chunk                   1
    Number of chunks              2
    Date/Time created             07/28/95 14:46:55
    DBSpace name                  rootdbs
    DBSpace owner                 informix


Validating Informix Dynamic Server reserved pages - PAGE_1PCHUNK & PAGE_2PCHUNK
          Using primary chunk page PAGE_2PCHUNK.

    Chunk number                  1
    Next chunk in DBSpace         2
    Chunk offset                  0
    Chunk size                    4000
    Number of free pages          1421
    DBSpace number                1
    Overhead                      0
    Flags                         40         Chunk is online
    Chunk name length             23
    Chunk path                    /home/server/root_chunk
.
.
.
```

Figure 29-38
oncheck -pr Output Showing Dbspace and Chunk Information

oncheck -pe

Execute **oncheck -pe** to obtain the physical layout of information in the chunk. 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 the extent of 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. Example output is shown in Figure 29-39 on page 29-58.

```
DBSpace Usage Report:  rootdbs              Owner:  informix  Created: 07/28/95

    Chunk: 1   /home/server/root_chunk                  Size     Used     Free
                                                        4000     2579     1421

        Disk usage for Chunk 1                           Start    Length
        ----------------------------------------  --------- ---------
        ROOT DBSpace RESERVED Pages                          0        12
        CHUNK FREE LIST PAGE                                12         1
        TBLSPACE TBLSPACE                                   13        50
        PHYSICAL LOG Pages                                  63       500
        LOGICAL LOG Pages - Log 1                          563       250
        LOGICAL LOG Pages - Log 2                          813       250
        LOGICAL LOG Pages - Log 3                         1063       250
        LOGICAL LOG Pages - Log 4                         1313       250
        LOGICAL LOG Pages - Log 5                         1563       250
        DATABASE TBLSPACE                                 1813         4
        sysmaster:informix.systables                      1817         8
        sysmaster:informix.syscolumns                     1825         8
        sysmaster:informix.sysindexes                     1833         8
        sysmaster:informix.systabauth                     1841         8
        sysmaster:informix.syscolauth                     1849         8
        sysmaster:informix.sysviews                       1857         8
        sysmaster:informix.sysusers                       1865         8
        sysmaster:informix.sysdepend                      1873         8
        sysmaster:informix.syssynonyms                    1881         8

    Chunk: 2   /home/server/raw_chunk                  Size     Used     Free
                                                        500         3      497

        Disk usage for Chunk 2                           Start    Length
        ----------------------------------------  --------- ---------
        OTHER RESERVED Pages                                 0         2
        CHUNK FREE LIST PAGE                                 2         1
        FREE                                                 3       497
```

Figure 29-39
oncheck -pe Output

UNIX

Using ON-Monitor

Select **Status→Spaces**. The first screen indicates whether the space is mirrored. An example of this output is shown in Figure 29-40.

```
Press ESC to return to the Status Menu.
Use arrow keys to move the cursor.
Press F3 or CTRL-B for chunk information on the highlighted dbspace/BLOBSpace.


                              DBSPACES/BLOBSPACES

                                        Number of     When
     Id         Name                    Chunks      Created      Status

      1         rootdbs                      1       08/05/95      N
      2         fstspace                     1       08/09/95      Y
       3          fstblob                          1        08/09/95       N
```

Figure 29-40
First Screen of ON-Monitor Spaces Option

The second screen indicates the mirror status of each chunk. An example is shown in Figure 29-41. These flags correspond to the flags displayed by **onstat -d**. For an explanation of the flag values, see "-d Option" on page 35-72.

```
Press ESC to return to the Status Menu.
Use arrow keys to move the cursor.


                          CHUNKS FOR cooked space

Chunk     Chunk       Pages       Pages  Full Pathname of Chunk           Status
 Id      Offset    In Chunk        Used

  2           0        2000         357  /home/server/test_chunk           PO-
  2           0        2000        2000  /home/server/test_mirr            MO-



     Dbspace has total of 2000 pages, 1643 of which are free
```

Figure 29-41
Second Screen of ON-Monitor Spaces Option

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 mirrored chunk is down
mis_recovering	Whether the mirrored 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 TbIspaces 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.

Using Command-Line Utilities

You can use the following command-line utilities to monitor tblspaces and extents.

oncheck -pt

Execute **oncheck -pt** with a database-name or table-name parameter to obtain the following information for each tblspace in the database or table:

- Number of extents
- Size of the first extent
- Size of the next extent
- Number of pages allocated
- Number of pages used

An example of the **oncheck -pt** output is shown in Figure 29-42. The table in the example is fragmented over multiple dbspaces. Because each fragment of a fragmented table resides in a separate tblspace, the **oncheck -pt** option always displays separate information for each fragment. The number of pages of table data in each fragment is displayed.

```
TBLspace Report for tpc:informix.account

                Table fragment in DBspace rootdbs

    Physical Address               100033
    Creation date                  03/31/97 13:25:21
    TBLspace Flags                 2          Row Locking
    Maximum row size               100
    Number of special columns      0
    Number of keys                 0
    Number of extents              2
    Current serial value           1
    First extent size              50
    Next extent size               25
    Number of pages allocated      2375
    Number of pages used           2370
    Number of data pages           2369
    Number of rows                 45001
    Partition partnum              2097154
    Partition lockid               2097154

    Extents
         Logical Page  Physical Page        Size
                    0         100ad5          50
                   50         100b2f        2325

                Table fragment in DBspace dbspace2

    Physical Address               200005
    Creation date                  03/31/97 13:25:21
    TBLspace Flags                 2          Row Locking
    Maximum row size               100
    Number of special columns      0
    Number of keys                 0
    Number of extents              1
    Current serial value           1
    First extent size              50
    Next extent size               25
    Number of pages allocated      550
    Number of pages used           528
    Number of data pages           527
    Number of rows                 10000
    Partition partnum              3145730
    Partition lockid               2097154

    Extents
         Logical Page  Physical Page        Size
                    0         200035         550
.
.
.
```

Figure 29-42
oncheck -pt Output

oncheck -pT

The **oncheck -pT** option returns *all* the information from the **oncheck -pt** option as well as the *additional* information shown in Figure 29-43. Each tblspace in the database or table that you supply is listed.

```
TBLSpace Usage Report for tpc:chrisw.account

    Type                 Pages      Empty  Semi-Full       Full  Very-Full
    ---------------- ---------- ---------- ---------- ---------- ----------
    Free                    20
    Bit-Map                  1
    Index                  471
    Data (Home)           3158
                     ----------
    Total Pages           3650

    Unused Space Summary

        Unused data slots                                    2
        Unused bytes per data page                          44
        Total unused bytes in data pages                138952

Index Usage Report for index iaccount on tpc:chrisw.account

                    Average    Average
    Level    Total No. Keys Free Bytes
    ----- -------- -------- ----------
        1        1        4       1973
        2        4      116        506
        3      466      128        217
    ----- -------- -------- ----------
    Total      471      128        223
```

Figure 29-43
oncheck -pT Output That Is Not Displayed by the oncheck -pt Option

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.

Using System Catalog Tables

Query the **sysfragments** table to obtain information about all tblspaces that hold a fragment. This table has a row for each tblspace that holds a table fragment or an index fragment. The **sysfragments** table includes the following columns.

Column	Description
fragtype	Table or index fragment
tabid	Table identifier
indexname	Index identifier
partn	Physical location (tblspace ID)
strategy	Distribution scheme (round-robin, expression, table-based index)
dbspace	Dbspacename for fragment
npused	Number of data pages or leaf pages
nrows	Number of rows or unique keys

Not all columns of **sysfragments** are documented in the preceding list. For a complete listing of columns, see the *Informix Guide to SQL: Reference*.

Monitoring TEXT and BYTE Data in a Blobspace

Monitor blobspaces to determine the available space and whether the blobpage size is optimal.

Using Command-Line Utilities

You can use the following command-line utilities to monitor TEXT and BYTE data in a blobspace.

onstat -d

The **onstat -d** option displays information on each blobspace and the chunks within each blobspace. Figure 29-44 shows an example of the output from this option. In this example, a blobspace called **fstblob** contains a single chunk called **blob_chunk**.

```
RSAM Version 7.30.UC1 -- On-Line -- Up 07:48:28 -- 4664 Kbytes

Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
40c980   1        1        1        1        N        informix rootdbs
40c9c4   2        1        2        1        N        informix fstdbs
40ca08   3        11       3        1        N B      informix fstblob
 3 active, 10 total

Chunks
address  chk/dbs offset   size     free    bpages   flags pathname
40c224   1   1   0        20000    14001            PO-   /home/server/root_chunk
40c2bc   2   2   0        2000     1659             PO-   /home/server/fst_chunk
40c354   3   3   0        4000    ~924     1000     POB   /home/server/blob_chunk
 3 active, 10 total
```

Figure 29-44
onstat -d Output with Information About Blobspaces

The **onstat -d** option lists the number of free blobpages in each blobspace chunk, as well as the number of total blobpages. The tilde (~) that precedes the `free` value indicates that this number is approximate. The number is approximate because the utility derives it from information stored in the disk version of the chunk free-map page, and not from the version stored in shared memory.

Another complication is that **onstat -d** 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 TEXT and BYTE objects and immediately execute **onstat -d**, the newly freed space does not appear in the **onstat** output.

onstat -O

The **onstat -O** option displays information about the staging-area blobspace and the Optical Subsystem memory cache. Figure 29-45 shows an example of the output from this option. 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**.

```
Informix Dynamic Server 7.30.UC1 -- On-Line -- Up 00:56:39 -- 10864 Kbytes

    Subsystem not available

Optical StageBlob Cache
System Cache Totals:                                  System Blob Totals:
Size    Alloc.  Avail.              Number  Kbytes    Number  Kbytes
0       0       0                   0       0         0       0

User Cache Totals:                                    User Blob Totals:
SID     User                 Size  Number  Kbytes    Number  Kbytes
No sessions currently using the cache
```

Figure 29-45
onstat -O Output

The first section of the display describes the following system-cache totals information:

size	is the size specified in the OPCACHEMAX configuration parameter.
alloc	is the number of 1-kilobyte pieces that the database server allocated to the cache.
avail	describes how much of **alloc** (in kilobytes) is not used.
number	is the number of TEXT and BYTE objects that the database server successfully put into the cache without overflowing.
kbytes	is the number of kilobytes of the TEXT and BYTE objects that the database server put into the cache without overflowing.
number	is the number of TEXT and BYTE objects that the database server wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of TEXT and BYTE 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 TEXT or BYTE 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:

SID	is the session ID for the user.
user	is the user ID of the client.
size	is the 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	is the number of TEXT and BYTE objects that the database server put into cache without overflowing.
kbytes	is the number of kilobytes of TEXT and BYTE objects that the database server put into the cache without overflowing.
number	is the number of TEXT and BYTE objects that the database server wrote to the staging-area blobspace.
kbytes	is the size of the TEXT and BYTE objects (in kilobytes) that the database server wrote to the staging-area blobspace.

oncheck -pB

This option gathers its data from the actual blobspace storage statistics.

Execute **oncheck -pB** with either a database name or a table name as a parameter. The display reports the following statistics:

- Number of blobpages used by this table or database in all blobspaces
- Blobpage fullness, by blobspace, for each blobspace in this table or database

The **oncheck** utility derives the number of free blobpages from the information stored in the shared-memory version of the chunk free-map page, not from the disk version. These statistics are the most current possible and might differ from the output of **onstat -d**, which is derived from the disk version of the free-map page. Example output is shown in Figure 29-46.

Figure 29-46
oncheck -pB Output

```
BLOBSpace Report for stores7:chrisw.catalog

    Total pages used by table         74

    BLOBSpace usage:

        Space          Page                          Percent Full
        Name           Number      Pages   0-25%  26-50%  51-75%  76-100%
        ------------------------------------------------------------------
        fstblob         0x300000        1    x
        fstblob         0x300001        1    x
        fstblob         0x300002        1    x
        fstblob         0x300003        1    x
        :
        fstblob         0x300046        1    x
        fstblob         0x300047        1    x
        fstblob         0x300048        1    x
        fstblob         0x300049        1    x
                                    ------
          Page Size is  8192           74
```

oncheck -pe

Execute **oncheck** with the **-pe** options for a detailed listing of chunk use; first the dbspaces are listed, then the blobspaces. The display provides the following blobspace-use information:

- 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

Tip: *The* ***oncheck -pe*** *option gives information on blobspace use in terms of database server pages, not blobpages.*

Example output is shown in Figure 29-47.

```
BLOBSpace Usage Report:  fstblob              Owner:  informix  Created: 03/01/97

     Chunk: 3   /home/server/blob_chunk                 Size     Used     Free
                                                      4000      304     3696

         Disk usage for Chunk 3                     Total Pages
         ------------------------------------------------------
         OVERHEAD                                             8
         stores7:chrisw.catalog                             296
         FREE                                              3696
```

Figure 29-47
oncheck -pe Output Showing Blobspace Use

UNIX

Using ON-Monitor

When you select the Spaces option of the Status menu, the database server displays a series of two screens. The first screen lists any blobspaces.

The second screen lists the following chunk information for each blobspace:

- Chunk ID
- Chunk pathname and offset
- Mirror status flags
- Pages in the chunk
- Number of used disk pages in the chunk

An example of this second screen is shown in Figure 29-41 on page 29-59.

Monitoring TEXT and BYTE Data in a Dbspace

You can monitor dbspaces to determine the number of dbspace pages that are used by TEXT and BYTE data.

Using Command-Line Utilities

Execute **oncheck** with the **-pT** options and either 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. Look for the blobpage type for information on TEXT and BYTE data. Example output is shown in Figure 29-48 on page 29-70.

The database server can store more than one TEXT or BYTE object on the same dbspace 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 TEXT or BYTE objects in the table.

```
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 29-48
oncheck -pT Output with a Tblspace Report That Contains TEXT or BYTE data

Monitoring High-Availability Data-Replication Status

Monitor the high-availability data-replication 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 high -availability data replication is on
- The values of the high-availability data-replication 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 high-availability data-replication pair, and in on-line mode:

```
RSAM Version 7.30.UC1     -- Dynamic Server (Prim) -- Up 45:08:57 -- 21604 Kbytes
```

This example shows a database server that is the secondary database server in a high-availability data-replication pair, and in read-only mode.

```
RSAM Version 7.30.UC1     -- Read-Only (Sec) -- Up 45:08:57 -- 21604 Kbytes
```

The following example shows a header for a database server that is not involved in high-availability data replication. The type for this database server is standard.

```
RSAM Version 7.30.UC1     -- Dynamic Server -- Up 20:10:57 -- 21604 Kbytes
```

onstat -g dri

To obtain full high-availability data-replication monitoring information, execute the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)
- The high-availability data-replication state (on or off)

- The paired database server
- The last high-availability data-replication checkpoint
- The values of the high-availability data-replication configuration parameters

Example output is shown in Figure 29-49. This example shows a primary database server, paired with a secondary database server that has the DBSERVERNAME of **beach_ol**. High-availability data replication has been started.

```
RSAM Version 7.30.UC1 -- On-Line (Prim) -- Up 00:01:23 -- 4584 Kbytes

Data Replication:
  Type         State        Paired server        Last DR CKPT (id/pg)
  standard      on          beach_ol                      25/8

  DRINTERVAL   30
  DRTIMEOUT    30
  DRAUTO       0
  DRLOSTFOUND  /usr/informix/etc/lost+found
```

Figure 29-49
onstat -g dri Output

oncheck -pr

If your database server is running high-availability data replication, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that high-availability data replication uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in Figure 29-50.

```
Validating Informix Dynamic 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 29-50
oncheck -pr PAGE_1ARCH Output for Database Server Running High-Availability Data Replication

UNIX

Using ON-Monitor

Use the Status menu, data-Replication option to see information about high-availability data replication. This option displays the same information as the **onstat -g dri** command-line option.

Using SMI Tables

The **sysdri** table, described in "sysdri" on page 34-17, contains the following columns.

Column	Description
type	High-availability data-replication server type
state	High-availability data-replication server state
name	Database server name
intvl	High-availability data-replication buffer flush interval
timeout	Network timeout
lostfound	High-availability data-replication lost+found pathname

Distributed Data

Section VIII

Chapter

30

Multiphase Commit Protocols

Two-Phase Commit Protocol 30-3
When Is the Two-Phase Commit Protocol Used? 30-3
What Goals Does the Two-Phase Commit Protocol Achieve? . . . 30-4
Two-Phase Commit Concepts 30-5
Phases of the Two-Phase Commit Protocol 30-6
Precommit Phase 30-7
Postdecision Phase 30-7
Examples of Two-Phase Commit Transactions 30-8
How the Two-Phase Commit Protocol Handles Failures. 30-10
What Types of Failures Does Automatic Recovery Handle?. 30-10
What Is the Administrator's Role in Automatic Recovery? . . 30-10
Automatic-Recovery Mechanisms for Coordinator Failure 30-10
Automatic-Recovery Mechanisms for Participant Failure . . . 30-14
Presumed-Abort Optimization 30-17
How Does Presumed-Abort Optimization Affect Automatic Recovery?. 30-18
Why Is an Optimization Realized? 30-18

Independent Actions 30-18
What Initiates Independent Action?. 30-19
Possible Results of Independent Action 30-19
Independent Actions That Allow Successful Completion of Transaction 30-20
Independent Actions That Result in an Error Condition . . . 30-20
Independent Actions That Result in Heuristic Decisions . . . 30-21
The Heuristic Rollback Scenario 30-22
Conditions That Result in a Heuristic Rollback 30-22
What Happens When a Heuristic Rollback Occurs? 30-23

The Heuristic End-Transaction Scenario 30-26
When to Perform a Heuristic End Transaction 30-26
How to Use onmode -Z 30-27
What Happens When the Transaction Is Ended Heuristically? 30-28
Tracking a Global Transaction 30-28

Two-Phase Commit Protocol Errors 30-29

Two-Phase Commit and Logical-Log Records 30-30
Logical-Log Records When the Transaction Commits 30-31
Logical-Log Records Written During a Heuristic Rollback 30-33
Logical-Log Records Written After a Heuristic End Transaction 30-35

Configuration Parameters Used in Two-Phase Commits 30-37
Function of the DEADLOCK_TIMEOUT Parameter 30-37
Function of the TXTIMEOUT Parameter 30-37

Heterogeneous Commit Protocol 30-38
Which Gateways Can Participate in a Heterogeneous Commit Transaction? 30-39
Enabling and Disabling Heterogeneous Commit 30-40
How Does Heterogeneous Commit Work 30-41
Precommit Phase 30-41
Gateway Commit Phase 30-41
Heterogeneous Commit Optimization. 30-42
Implications of a Failed Heterogeneous Commit 30-43
Database Server Coordinator Failure 30-43
Participant Failure 30-44
Interpreting Heterogeneous Commit Error Messages 30-45

A multiphase commit protocol is a mechanism that the database server uses to process transactions that span multiple database servers. Understanding how multiphase commit protocols work becomes important only when a deviation from one of these protocols occurs. The database server supports two multiphase commit protocols: two-phase commit protocol and heterogeneous commit protocol.

For information on recovering manually from a failed two-phase commit, see Chapter 31, "Recovering Manually from Failed Two-Phase Commit."

Two-Phase Commit Protocol

The two-phase commit protocol governs the order in which a two-phase commit transaction is performed and provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction.

When Is the Two-Phase Commit Protocol Used?

A database server automatically uses the two-phase commit protocol for any transaction that performs modifications to data on more than one database server.

For example, consider the configuration shown in Figure 30-1, which includes three database servers.

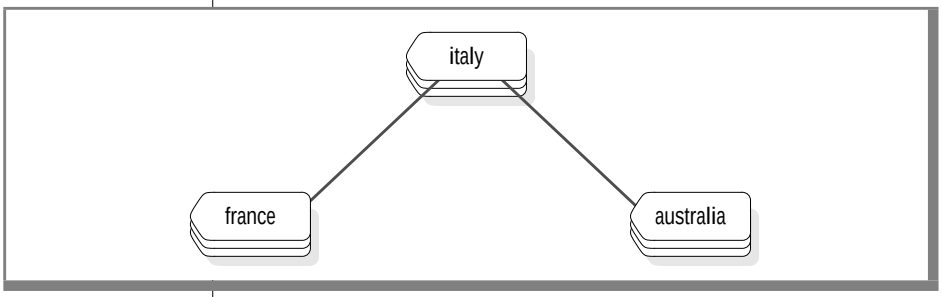


Figure 30-1
Connected Database Servers

If you execute the commands shown in Figure 30-2, the result is one update and two inserts at three different database servers.

```
CONNECT TO stores7@italy
BEGIN WORK
    UPDATE stores7:manufact SET manu_code = 'SHM' WHERE manu_name = 'Shimara'
    INSERT INTO stores7@france:manufact VALUES ('SHM', 'Shimara', '30')
    INSERT INTO stores7@australia:manufact VALUES ('SHM', 'Shimara', '30')
COMMIT WORK
```

Figure 30-2
Example of a Distributed Transaction

The database server automatically uses the two-phase commit protocol for this transaction.

What Goals Does the Two-Phase Commit Protocol Achieve?

Distributed transactions that include multiserver modifications use the two-phase commit protocol to achieve the following two goals:

- Ensure that all participating database servers receive the same instruction, either to commit or to roll back a transaction
- Ensure that all participating database servers implement the same action (either a commit or a rollback), regardless of local or network failures during the protocol

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. The database server uses the two-phase commit protocol to coordinate the work performed at multiple database servers on behalf of a single transaction.

Two-Phase Commit Concepts

Every transaction that uses the two-phase commit protocol has a *coordinator* and one or more *participants*:

- Coordinator

  The two-phase commit protocol always assigns the role of coordinator to the *current* database server. In other words, the database server that is managing the current database when the transaction is processed is the coordinator. In the example transaction given in Figure 30-2 on page 30-4, 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 stores7@france
  ```

  The role of coordinator cannot change during a single transaction. The coordinator for a distributed transaction is displayed by the **onstat -x** option. For an example of **onstat -x** output, see "Monitoring Transactions" on page 29-41.

  In the two-phase commit protocol, the transaction that is under the direction of the coordinator is called the *global transaction*.

- Participants

  The database servers that perform operations under the direction of the coordinator are the participants. In Figure 30-1 on page 30-4, the participants are **france** and **australia**. The work that each participant database server performs is a *piece of work* associated with the global transaction. In this example, the coordinator database server, **italy**, also functions as a participant because it is also doing a piece of work, which is the update.

The two-phase commit protocol relies on two kinds of communication, messages and logical-log records:

- Messages

  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

  Logical-log records of the transaction are kept on stable storage (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 about the logical-log records that are written during two-phase commit protocol, refer to "Two-Phase Commit and Logical-Log Records" on page 30-30.

Phases of the Two-Phase Commit Protocol

In a two-phase commit transaction, all the instructions are first sent to all the participants to perform data modifications (for example, inserts). After the coordinator sends these instructions, it 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**: If the data modifications satisfy all deferred constraints, the participants return messages to the coordinator indicating that their piece of work can be committed. If a modification does not satisfy a constraint, the participant returns a negative response to the coordinator.
3. **Coordinator**: The coordinator determines whether to commit or roll back the transaction. If at least one participant is unable to perform the modifications, the coordinator makes the decision to roll back the 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 send 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.

Examples of Two-Phase Commit Transactions

Figure 30-3 represents a two-phase commit protocol that results in a committed transaction.

The postdecision phase begins at the instant when the coordinator records its decision to commit or roll back. In this case, phase two begins when the coordinator writes the commit work logical-log record to disk.

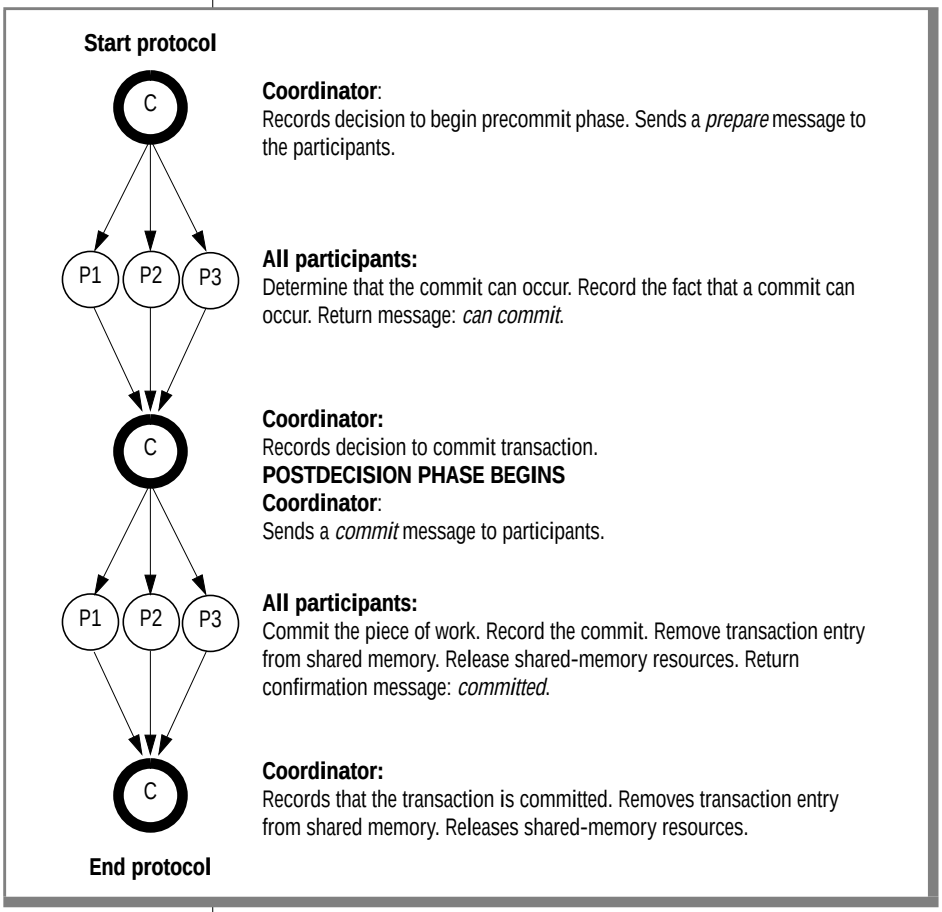


Figure 30-3
Two-Phase Commit Protocol That Results in a Committed Transaction

Figure 30-4 represents a two-phase commit that results in a rolled-back transaction. For some reason, probably a deferred constraint violation, the P1 participant determines that the commit cannot occur and returns a `cannot commit` message to the coordinator.

The participants do not send a confirmation to the coordinator when the piece of work is rolled back. The coordinator does not record a completed transaction. In this example, the postdecision phase begins when the coordinator writes the rollback work logical-log record to disk.

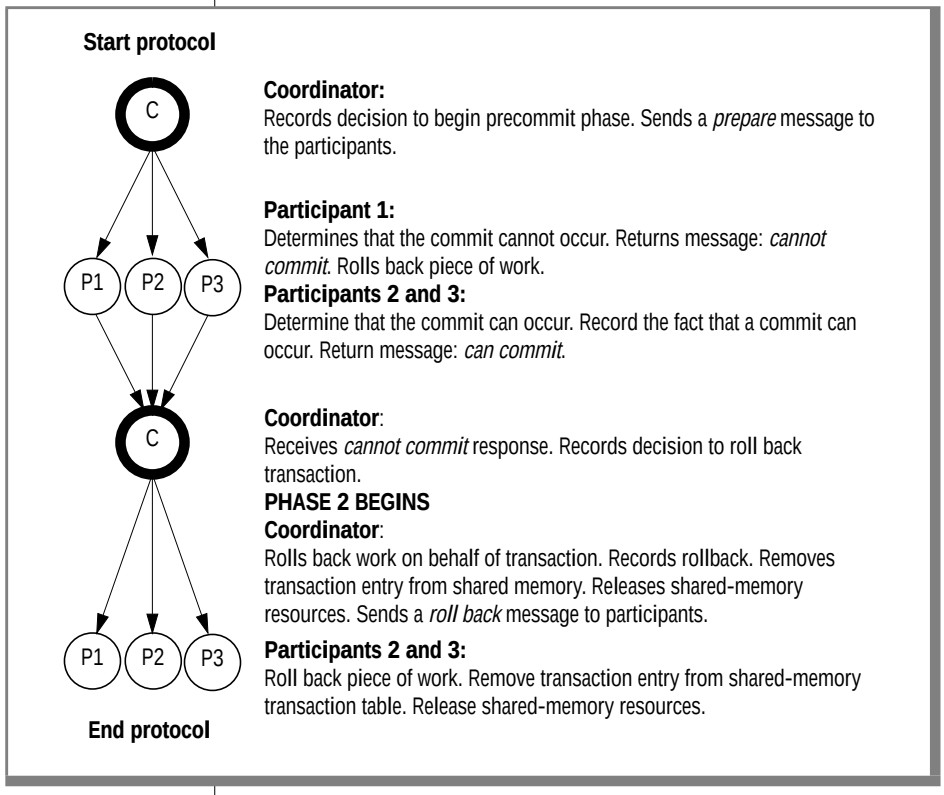


Figure 30-4
A Successful Two-Phase Commit Where Transaction Rolls Back

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.

What Types of Failures Does Automatic Recovery Handle?

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

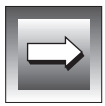
What Is the Administrator's Role in Automatic Recovery?

The only role of the administrator in automatic recovery is to bring the coordinator or participant (or both) back on-line after a system or network failure.

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-abort optimization. (See "Presumed-Abort Optimization" on page 30-17.)

The coordinator must initiate recovery if the coordinating thread fails after a decision to commit the transaction but before the two-phase commit protocol is complete.



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.*

Coordinator Fails Before Decision to Commit Transaction

If the coordinator fails before it decides whether to commit the transaction, the recovery mechanism is as follows. First, the system administrator must bring the coordinator back on-line. Next, the fast-recovery mechanism rolls back the global transaction on the coordinator. Meanwhile, the participant waits for the time period specified by the configuration parameter TXTIMEOUT for a message from the coordinator to either commit or roll back its piece of work. After waiting the specified time, the participant attempts to fork a new thread on the coordinator to determine the transaction status. (The participant will not be able to fork this new thread until the coordinator is back on-line). Because the coordinator rolled back the global transaction, no transaction is in shared memory. The presumed-abort optimization is in effect, and the participant rolls back its piece of work.

Coordinator Fails After Decision to Roll Back Transaction

If the coordinator fails after deciding to roll back the transaction, but before completing the two-phase commit protocol, the recovery mechanism is as follows. First, the system administrator must bring the coordinator back on-line. The global transaction at the coordinator has already been removed from memory, so the coordinator takes no further action. Meanwhile, the participant waits for the time period specified by the configuration parameter TXTIMEOUT for a message from the coordinator to either commit or roll back its piece of work. After waiting the specified time, the participant attempts to fork a new thread on the coordinator to determine the transaction status. (The participant will not be able to fork this new thread until the coordinator is back on-line). Because the coordinator rolled back the global transaction, no transaction is in shared memory. The presumed-abort optimization is in effect, and the participant rolls back its piece of work.

Coordinator Fails After Decision to Commit Transaction

If the coordinating thread completes the precommit phase, makes a decision to commit the transaction, but is terminated before the two-phase commit protocol can be completed, the coordinator-recovery mechanism goes into effect.

First, if a system failure occurs, the administrator must reinitialize shared memory and restart the database server; otherwise, coordinator recovery begins as part of the database server processing.

As part of its regular activity, the **main_loop** thread at the coordinator database server detects the following situations:

1. A two-phase commit protocol was under way.
2. The coordinating thread had reached a decision to commit. (Whether the coordinator sent messages to the participants to commit the transaction is ambiguous.)
3. The coordinating thread was terminated prematurely.

To complete the transaction, the **main_loop** thread forks a new coordinating thread. This new coordinating thread establishes a new connection at each participant database server.

The new coordinating thread sends a message to each participant thread to determine the status of the piece of work that was assigned to that participant database server. If the participant received a commit message from the coordinator before it failed, the participant would have committed the piece of work and removed the entry from its transaction table but would have no knowledge of the transaction. Thus, if the new coordinating thread receives a message that the transaction status is unknown, it assumes that the piece of work was already committed.

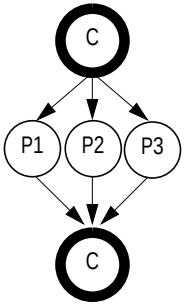
If the new coordinating thread receives a message that the transaction status is `can commit`, it sends a message to the participant to commit the transaction. After all participants send acknowledgments indicating that their piece of work is committed, the coordinating thread ends the transaction.

If the coordinating thread cannot contact one or more of the participants, or if execution errors are detected, messages are recorded in the database server message log.

The coordinating thread then continues its attempts to contact all participants and to complete the protocol. Figure 30-5 illustrates the coordinator-recovery process.

Figure 30-5
Example of Automatic Coordinator Recovery

Start protocol



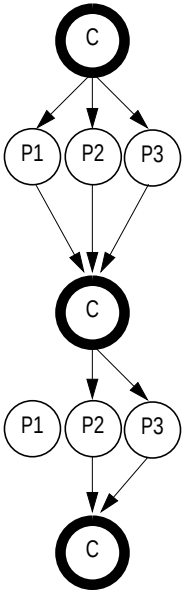
Coordinator:
Records decision to begin precommit phase. Sends a *prepare* message to the participants.

All Participants:
Determine that the commit can occur. Record the fact that a commit can occur. Return message: *can commit*.

Coordinator:
Records decision to commit transaction.
Coordinator Fails

Start coordinator recovery

After the coordinator returns on-line, the main_loop thread forks a new coordinating thread.



Coordinator:
Queries each participant for transaction status.

Participant 1:
Reports *unknown status*.
Participants 2 and 3:
Determine that the commit can occur. Record the fact that the commit can occur. Return message *can commit*.

Coordinator:
Sends a *commit* message to participants 2 and 3. Participant 1 is assumed to have committed.

Participants 2 and 3:
Commit the piece of work. Record the commit. Remove transaction entry from shared-memory transaction table. Release shared-memory resources. Return confirmation message: *committed*.

Coordinator:
Records that the transaction is committed. Removes transaction entry from shared memory. Releases shared-memory resources.

End coordinator recovery

Automatic-Recovery Mechanisms for Participant Failure

Participant recovery occurs whenever a participant thread precommits a piece of work but 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.

The actual participant-recovery mechanism depends on whether the global transaction was committed or rolled back.

Participant Fails—Global Transaction Committed

In this scenario, the participant thread precommits its piece of work successfully and then unexpectedly terminates. The coordinator then decides to commit the global transaction. When the coordinator sends the terminated participant a message to commit, it receives an error indicating that the participant is down. At this point, the coordinator takes the following actions:

1. Drops the existing connection to the participant
2. Forks a new participant thread
3. Resends the message to commit the piece of work

If the participant is down, the coordinator is not able to fork a new participant thread. The coordinator continues to loop through this procedure, however, until it succeeds (when the system administrator brings the participant back on-line).

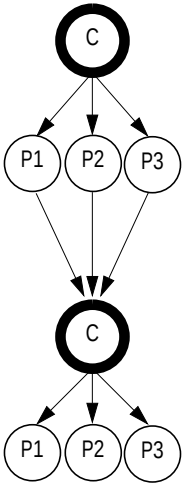
Participant Fails—Global Transaction Rolled Back

In this scenario, the participant thread precommits its piece of work successfully and then unexpectedly terminates. The coordinator then determines to roll back the global transaction. The coordinator sends a message to the terminated participant to roll back its piece of work. The participant does not receive the message because it is not running, but the coordinator in this scenario closes the global transaction anyway because a coordinator never waits for an acknowledgment on a rollback transaction.

When the system administrator brings the participant back up, the piece of work from the global transaction is still waiting for a message from the coordinator to commit or roll back. The participant thread, which is running the piece of work, waits for a period of time specified by the TXTIMEOUT variable and then forks a new participant thread. This new participant thread is responsible for determining the state of the global transaction. It forks a new thread on the coordinator and sends a message to this thread to determine the status of the global transaction. Because the coordinator previously made the decision to roll back the global transaction, no transaction status is in the coordinator shared memory. The participant thread rolls back its piece of work, therefore, and recovery is complete. Figure 30-6 on page 30-16 illustrates this scenario.

Figure 30-6
Automatic Participant Recovery

Start protocol



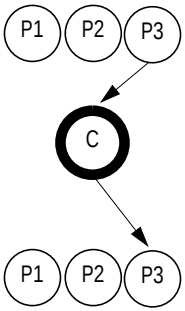
Coordinator:
Records decision to begin precommit phase. Sends a *prepare* message to the participants.

Participant 1:
Returns message: *cannot commit*.
Participants 2 and 3:
Return message: *can commit*.
Participant 3 Fails

Coordinator:
Receives *cannot commit* response. Records decision to roll back transaction. Rolls back work on behalf of transaction. Removes transaction entry from shared memory. Releases shared-memory resources. Sends a *rollback* message to each participant.

Start participant recovery

After Participant 3 returns on-line, the transaction on P3 waits for a message from the coordinator until TXTIMEOUT elapses. When TXTIMEOUT elapses, the participant forks a new coordinating thread.



Participant 3:
Queries coordinator about the transaction status.

Coordinator:
Reports that transaction status is unknown.

Participant 3:
Interprets status as instruction to roll back. Rolls back piece of work. Removes transaction entry from shared memory. Releases shared-memory resources.

End participant recovery

Race Conditions

A race condition might develop between the coordinator and the participant when the global transaction is committed.

For example, a race condition can result if a network failure occurs after the coordinator determines that the transaction should be committed. When the network comes back up, the coordinator tries to send the commit instruction to all the participants as described in "Participant Fails—Global Transaction Committed" on page 30-14. If a participant does not receive a message from the coordinator before the TXTIMEOUT period, this participant attempts to contact the coordinator using the mechanism in "Participant Fails—Global Transaction Rolled Back" on page 30-15. In this scenario, however, the global transaction is committed, so the participant finds a transaction status in the coordinator shared memory. In this case, the participant goes to sleep, allowing the coordinator time to send it the message to commit its piece of work.

Presumed-Abort Optimization

Presumed-abort optimization is the term that describes how the two-phase commit protocol handles the rollback of a transaction (an abort).

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.

How Does Presumed-Abort Optimization Affect Automatic Recovery?

Each participant database server must initiate automatic recovery if the coordinating thread fails in the following situations:

- Before it makes a decision to commit or roll back the transaction
- After it decides to roll back a transaction

This responsibility is part of the presumed-abort optimization. (See "Automatic-Recovery Mechanisms for Coordinator Failure" on page 30-10.)

Why Is an Optimization Realized?

Optimization is realized because the coordinator is not required to flush the logical-log record (BEGPREP) that indicates a two-phase commit protocol has begun. If this information is lost (for example, if the coordinator fails before making a decision), each participant automatically rolls back its piece of work. Thus the logical log can be buffered, which represents the most significant part of the streamlined processing.

To a lesser extent, message traffic is reduced because the coordinator receives acknowledgment only when a transaction commits. Participants do not acknowledge rollbacks.

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 31, "Recovering Manually from Failed Two-Phase Commit.")

What Initiates 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 kills a participant thread during the postdecision phase of the protocol using **onmode -z**.
- An administrator kills a participant transaction (piece of work) during the postdecision phase of the protocol using **onmode -Z**.
- An administrator kills a global transaction at the coordinator database server using **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 Successful Completion of Transaction

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** (kill the coordinator thread) or **onmode -Z** (kill 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.

Killing 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 on-line 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 30-22)
- Heuristic end transaction (described in "The Heuristic End-Transaction Scenario" on page 30-26).

Once 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

Heuristic rollback is an independent action that either the database server or the administrator can take to roll 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 Chapter 33, "Configuration Parameters.") 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 kill 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-Water Mark

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-water mark (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-water mark 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 kill 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.)

What Happens When a Heuristic Rollback Occurs?

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 shared-memory transaction table.)
4. The coordinator writes an ENDTRANS record in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)
5. The coordinator returns error -699 to the application, as follows:

   ```
   -699 Transaction heuristically rolled back.
   ```

6. 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 shared-memory 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 "Determine If a Transaction Was Implemented Inconsistently" on page 31-4.)

The Heuristic End-Transaction Scenario

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 database server shared-memory 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 on-line 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-water mark.

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-water mark (LTXEHWM). If this point is reached, normal processing is suspended. At some point after the LTXEHWM high-water mark 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 kill 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, refer to "Kill a Database Server Transaction" on page 35-34.

The *address* parameter is obtained from **onstat -x** output. See "-x Option" on page 35-92.

What Happens 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:

```
(timestamp) 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.

Tracking a Global Transaction

You can use the **onstat -x** utility to track transactions as they execute. For sample output of the **onstat -x** utility, refer to "Monitoring Transactions" on page 29-41. The fields displayed by **onstat -x** are described in detail in "-x Option" on page 35-92.

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 "What Happens When a Heuristic Rollback Occurs?" on page 30-23. 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 "What Happens When a Heuristic Rollback Occurs?" on page 30-23. 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 30-20.

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 31, "Recovering Manually from Failed Two-Phase Commit.")

The following logical-log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

For information about these logical-log records, see "Logical-Log Record Types and Additional Columns" on page 37-7.

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 30-7 illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.

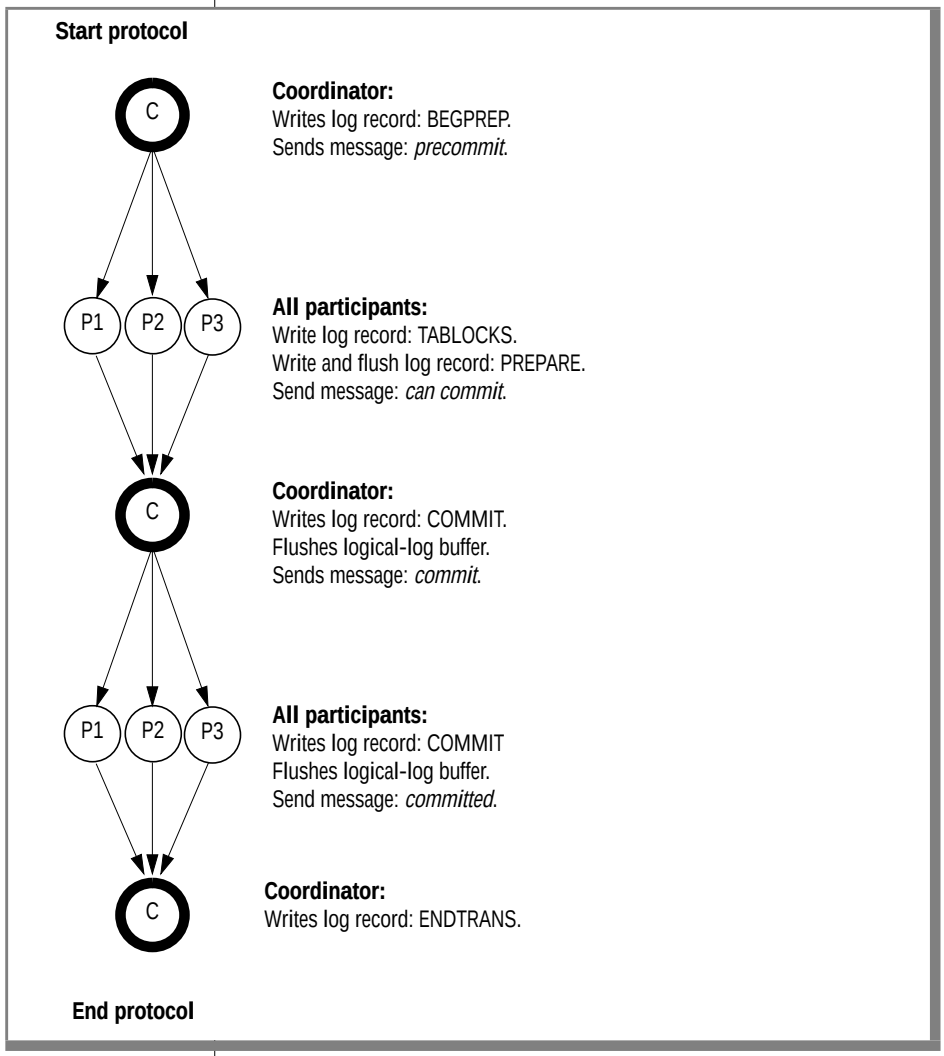


Figure 30-7
Logical-Log Records Written During a Committed Transaction

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 30-7 on page 30-31 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 30-8 on page 30-34 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 30-7 on page 30-31. 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 30-22.

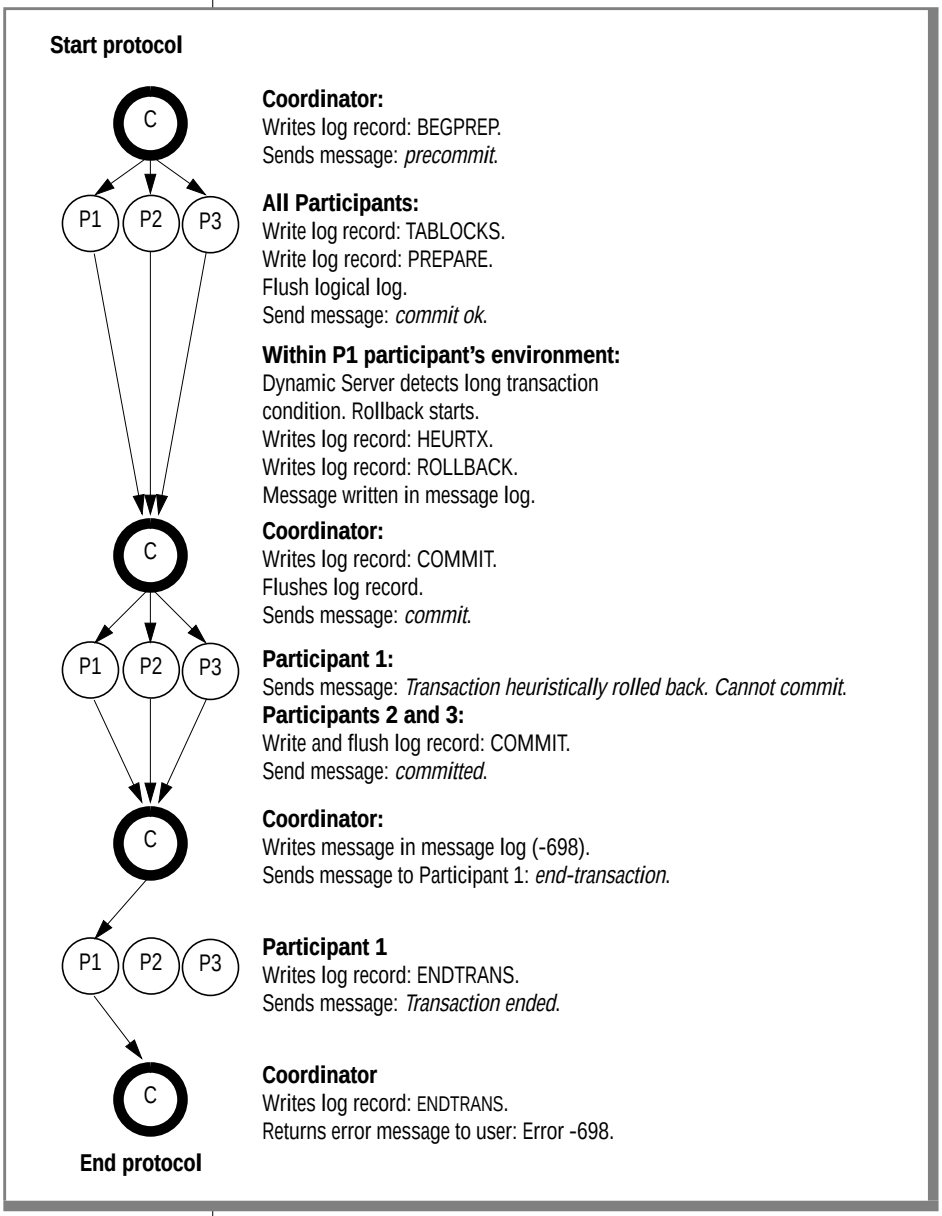


Figure 30-8
Logical-Log Records Written During a Heuristic Rollback

Logical-Log Records Written After a Heuristic End Transaction

Figure 30-9 on page 30-36 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 killing a transaction (see "Kill a Database Server Transaction" on page 35-34) at a participant database server after the participant has sent a `can commit` message. In Figure 30-9, the heuristic end transaction is assumed to have occurred at the P1 participant. The result is an inconsistently implemented transaction. See "The Heuristic End-Transaction Scenario" on page 30-26.

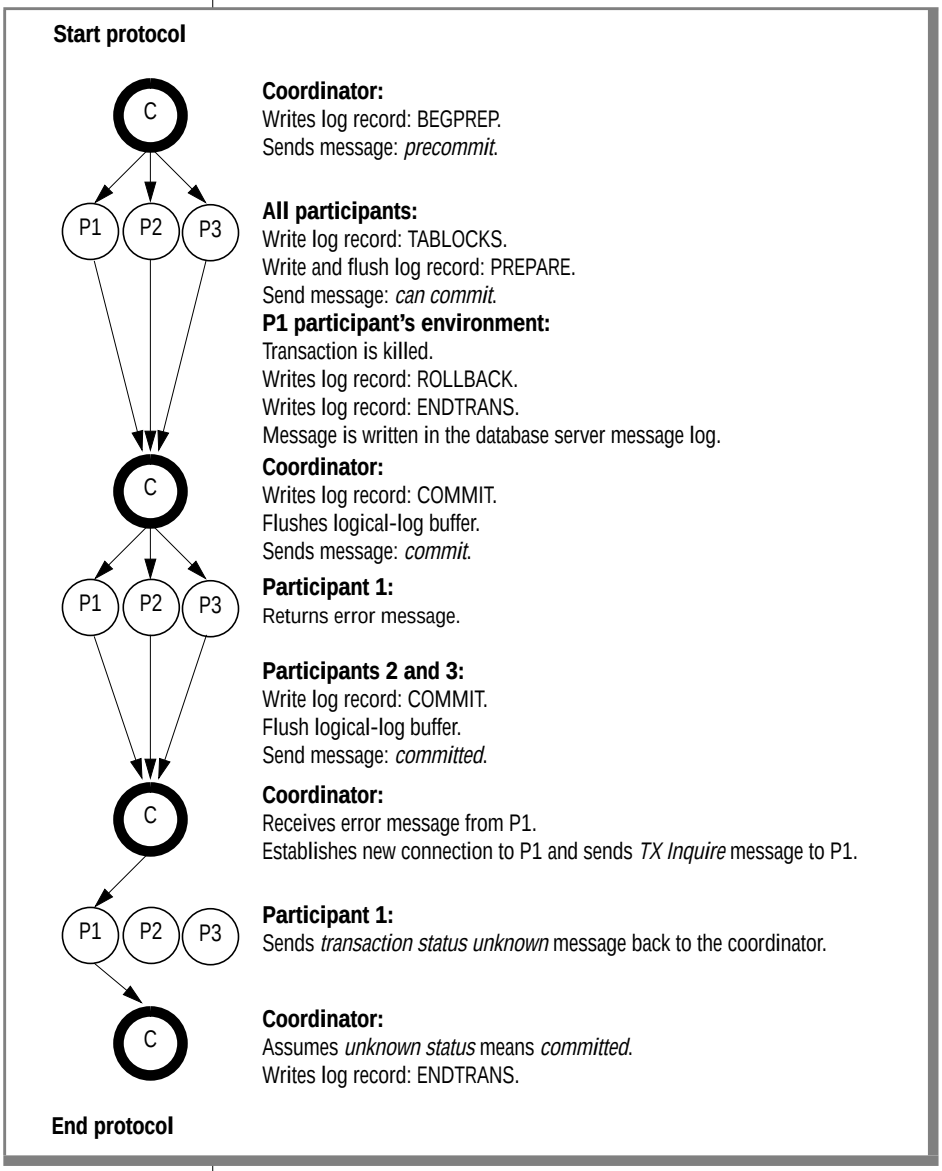


Figure 30-9
Logical-Log Records Written During a Heuristic End Transaction

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.

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 abort 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 30-10.

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 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 Informix database 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 30-10 illustrates this scenario.

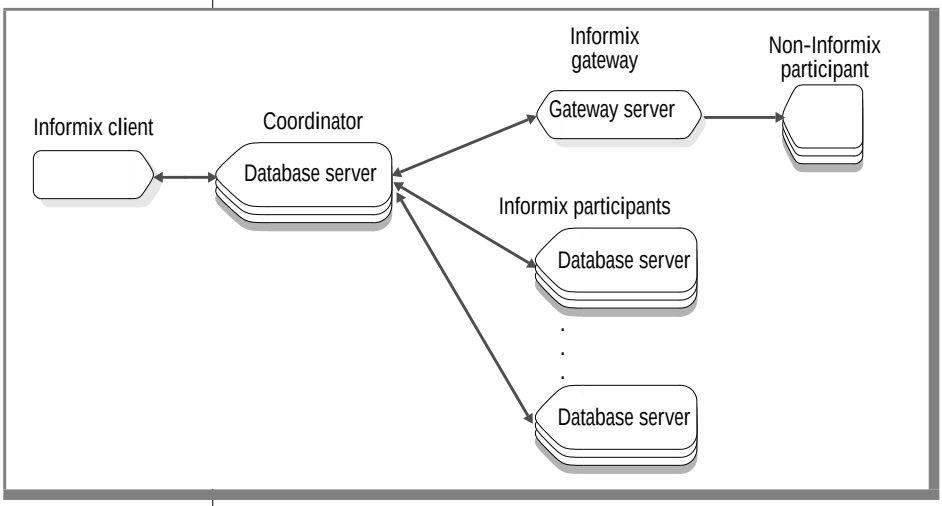


Figure 30-10
Configuration That Requires Heterogeneous Commit for Distributed Transactions

Which Gateways Can Participate in a Heterogeneous Commit Transaction?

An Informix gateway acts as a bridge between an Informix application (in this case, a database server) and a non-Informix database server. An Informix gateway allows you to use an 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.

Gateway	Database Servers
INFORMIX-Enterprise Gateway with DRDA	IBM DB2, OS / 400, SQL / DS
INFORMIX-Enterprise Gateway for EDA / SQL	EDA / SQL
INFORMIX-Enterprise Gateway Manager	Any database server with ODBC connectivity

Enabling and Disabling Heterogeneous Commit

Use one of the following methods to enable or disable heterogeneous commit:

UNIX

- Use ON-Monitor. ♦
- Change the HETERO_COMMIT configuration parameter in your configuration file.

In either case, the change does not take effect until you reinitialize shared memory by bringing the database server off-line and then on-line again.

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, heterogeneous commit or two-phase commit, the transaction coordinator uses 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 Does Heterogeneous Commit Work

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 30-7.

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 30-11 on page 30-42 illustrates this process.

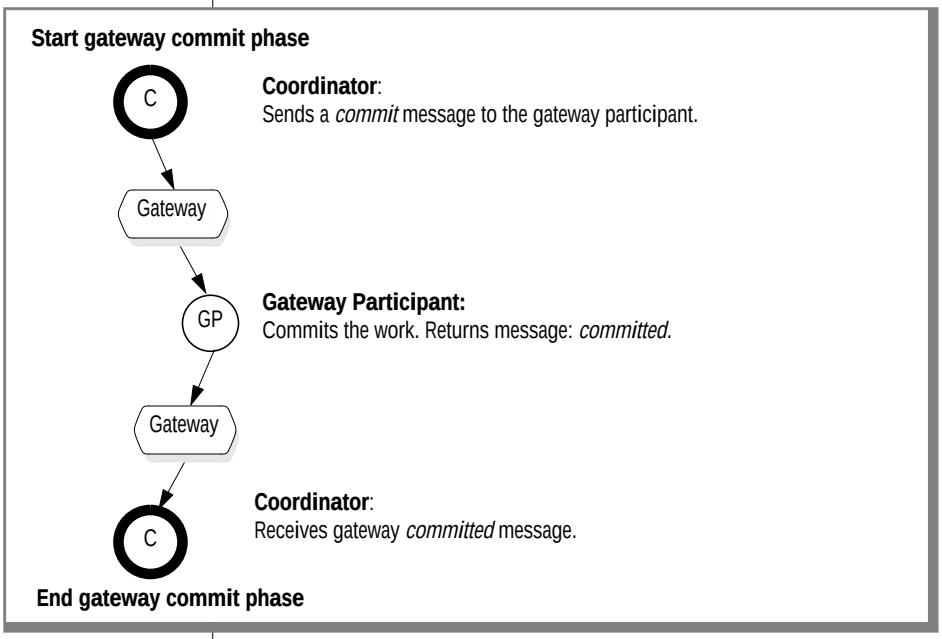


Figure 30-11
Heterogeneous Commit Phase That Results in a Committed Transaction

If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction, as Figure 30-11 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 aborted 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 aborted upon recovery. This abort might 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" on page 30-44.

Interpreting 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

31

Recovering Manually from Failed Two-Phase Commit

Procedure to Determine If Manual Recovery Is Required 31-3
Determine If a Transaction Was Implemented Inconsistently . . . 31-4
Global Transaction Killed Prematurely 31-4
Heuristic End Transaction 31-4
Heuristic Rollback 31-5
Determine If the Distributed Database Contains Inconsistent Data 31-6
Obtain Information from the Logical Log 31-7
The Global Transaction Identifier 31-8
Decide If Action Is Needed to Correct the Situation 31-9

Example of Manual Recovery 31-10

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 30-18.) In these situations, it might be necessary to recover manually from the transaction.

This chapter describes the following topics:

- How to determine if you need to recover manually from an inconsistently implemented two-phase commit transaction
- How to perform a manual recovery

Procedure to Determine 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:

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.

Determine 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 Killed Prematurely

If you executed an **onmode -z** command to kill 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 30-20.) 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 30-26.) 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.

Determine 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, here are three possible reasons:

- 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.

Obtain 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 the 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 "The Global Transaction Identifier" on page 31-8.
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.

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 31-1 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 00000000 00000000 00000000 ........ ........
         00000000 00000001 6a756469 74685f73 ........ judith_s
         6f630000 736f6374 63700000          oc..soct cp..
```

Figure 31-1 *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 31-2 with that of the BEGPREP record in Figure 31-1.

```
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 31-2
Output of the onlog -l Option for a PREPARE Record

Decide 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    xididlink
.....
17018    16CKPOINT  0       0  13018    0
18018    20BEGIN    2       1  0        08/27/91 10:56:57 3482     nhowe
1802c    32HINSERT  2       0  18018    1000018  102      4
1804c    40CKPOINT  0       0  17018    1
    begin    xid id addr     user
    1        2  1  1802c    nhowe
19018    72BEGPREP  2       0  1802c    6d69    1
19060    16COMMIT   2       0  19018    08/27/91 11:01:38
1a018    16ENDTRANS 2       0  19060    580543
```

The following excerpt is taken from the logical log at the database server **apex**:

```
addr    len type    xididlink
.....
16018    20BEGIN    2 10        08/27/91 10:57:07 3483     pault
1602c    32HINSERT  2       0  16018    1000018  102      4
1604c    68PREPARE  2       0  1602c    eh
17018    16HEURTX   2       0  1604c          1
17028    12CLR      2       0  1602c
17034    16ROLLBACK 2       0  17018    08/27/91 11:01:22
17044    40CKPOINT  0       0  15018    1
       begin    xid      id addr     user
       1        2        1  17034    --------
18018    16ENDTRANS 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 example situation, you would take the following steps:

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
```

Reference

Section IX

Chapter

32

ON-Monitor for UNIX

Using ON-Monitor 32-3
Help and Navigation Within ON-Monitor 32-4
Executing Shell Commands from Within ON-Monitor 32-4

ON-Monitor Screen Options 32-4

Setting Configuration Parameters with ON-Monitor 32-12

This section is a quick reference for the database server ON-Monitor screens. You can use it to determine the purpose and use of a specific screen or option.

WIN NT

ON-Monitor is not available on Windows NT. ♦

Using ON-Monitor

To start ON-Monitor, execute the following command from the operating-system prompt:

```
% onmonitor
```

If you are logged in as user **informix** or user **root**, the main menu appears. All users other than **informix** and **root** have access only to the Status menu.

The ON-Monitor main menu displays six additional menus, the Force-Ckpt option, and the Exit option. The six additional menus are as follows:

- Status menu
- Parameters menu
- Dbspaces menu
- Mode menu
- Archive menu
- Logical-Logs menu

The main menu, the six additional menus, and the Force-Ckpt option are shown on the following pages (Figure 32-1 through Figure 32-7).

Help and Navigation Within ON-Monitor

All menus and screens in ON-Monitor function in the same way. For menus, use the arrow keys or SPACEBAR to scroll to the option you want to execute and press RETURN, or press the first capitalized letter of the option (usually the first letter). When you move from one option to the next by pressing SPACEBAR or an arrow key, the option explanation (line 2 of the menu) changes.

If you want general instructions for a specific screen, press CTRL-W. If you need help to determine what you should enter in a field on the screen, use the TAB key to highlight the field and press CTRL-F or F2.

Some of the menus display ellipses (...) on the far right or left side. The ellipses indicate that you can move in the direction of the dots, using the arrow keys or SPACEBAR, to view other options.

Executing Shell Commands from Within ON-Monitor

To execute a shell command from within ON-Monitor, type an exclamation point (!) followed by the command. For example, to list the files in the current directory, type the following command:

```
!ls
```

ON-Monitor Screen Options

The following pages show the options that are available from the different ON-Monitor menus, and what the options do.

Figure 32-1
Status Menu

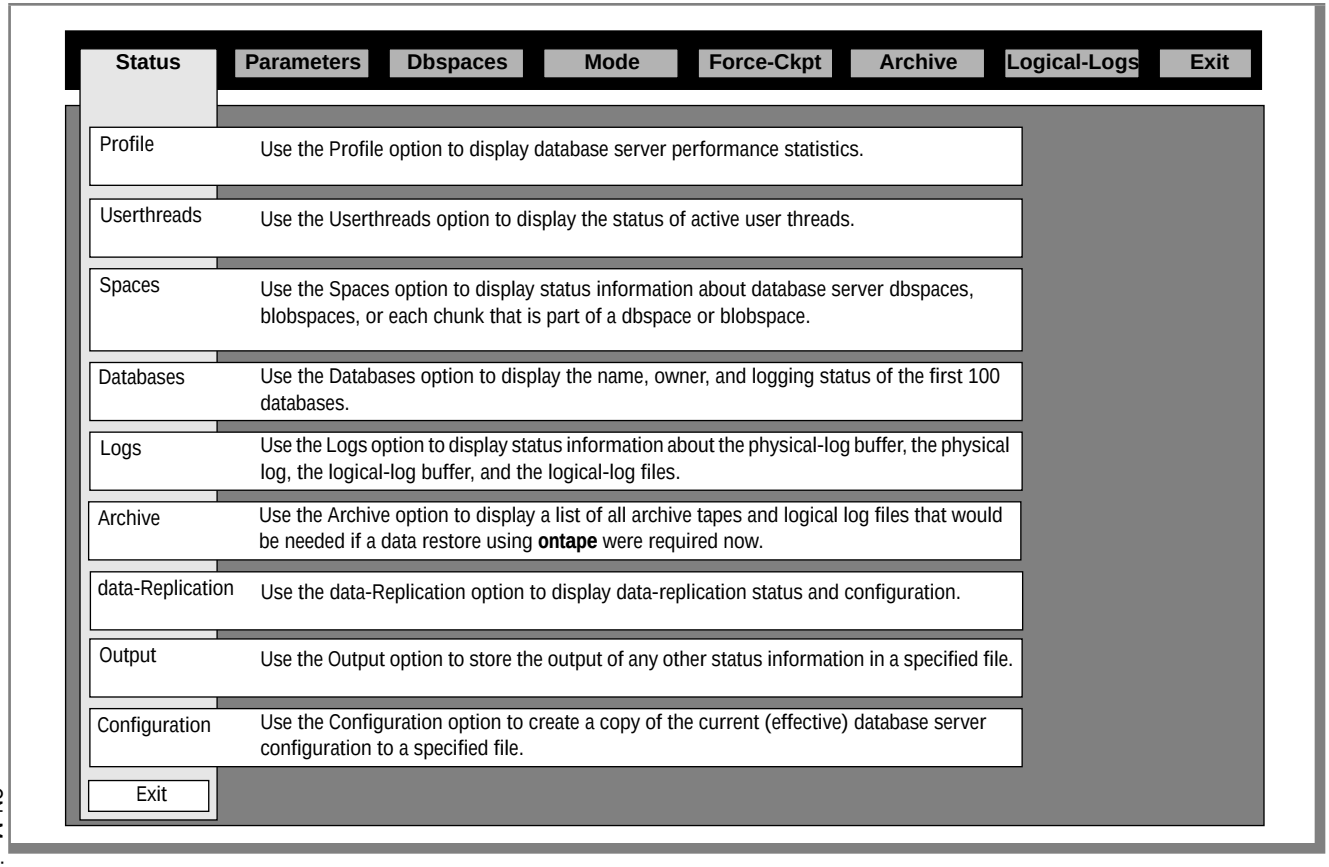


Figure 32-2
Parameters Menu

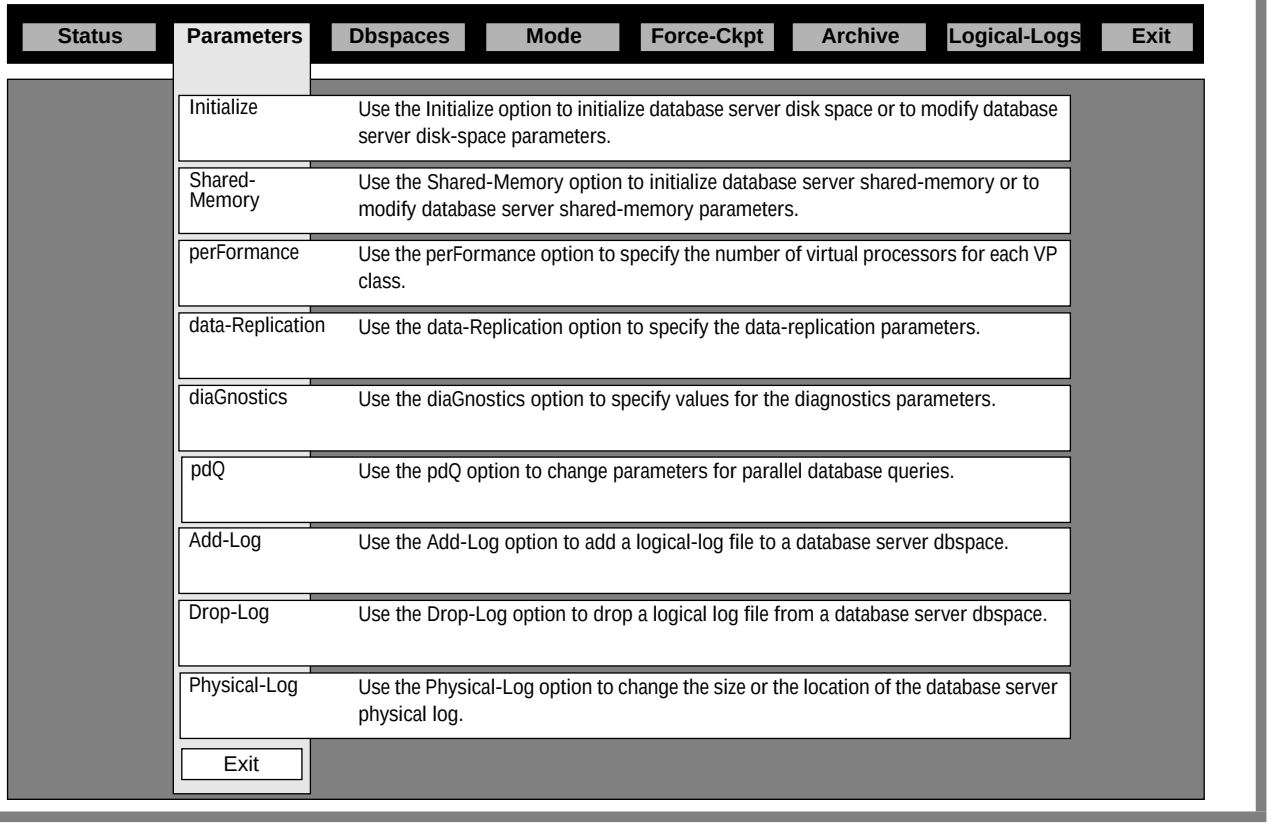


Figure 32-3
Dbspaces Menu

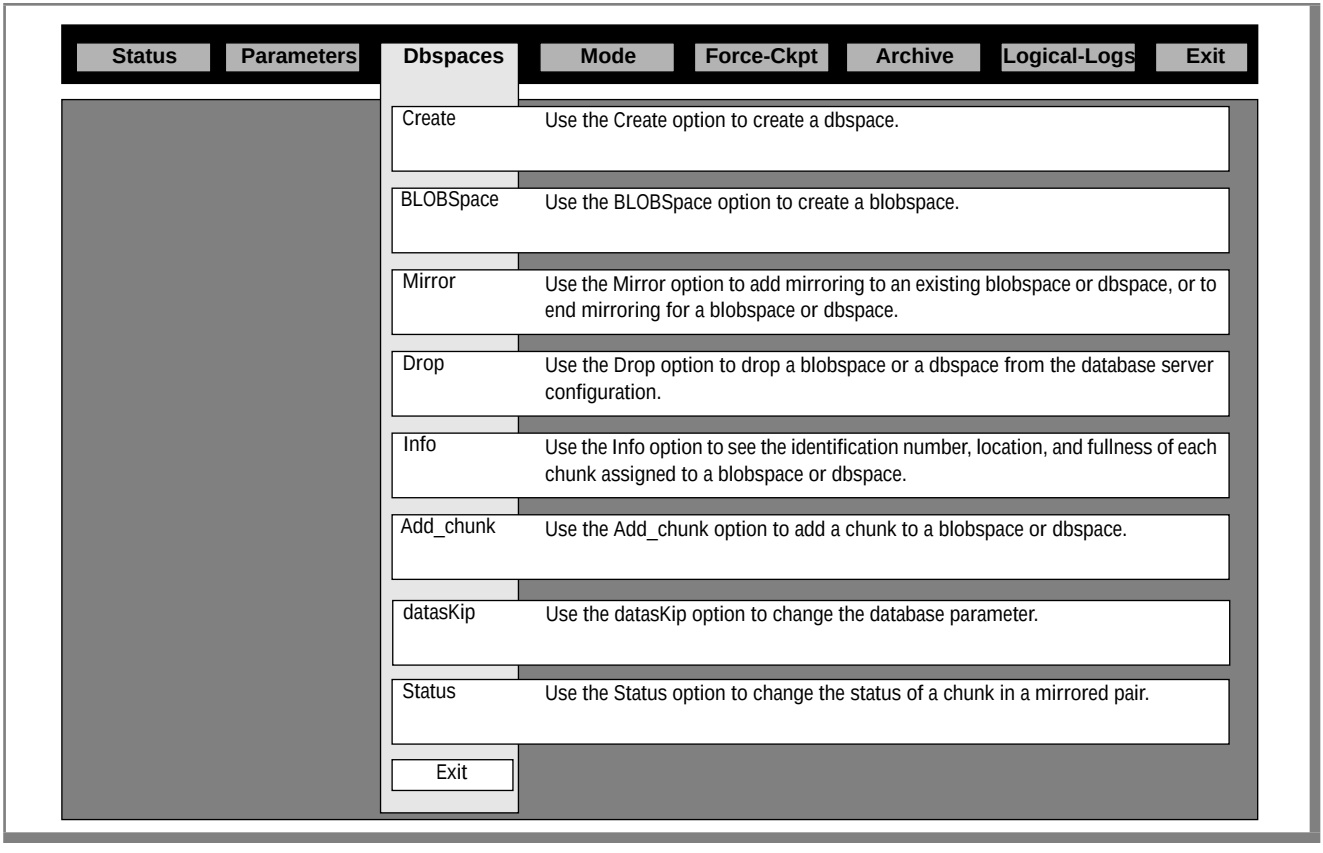


Figure 32-4
Mode Menu

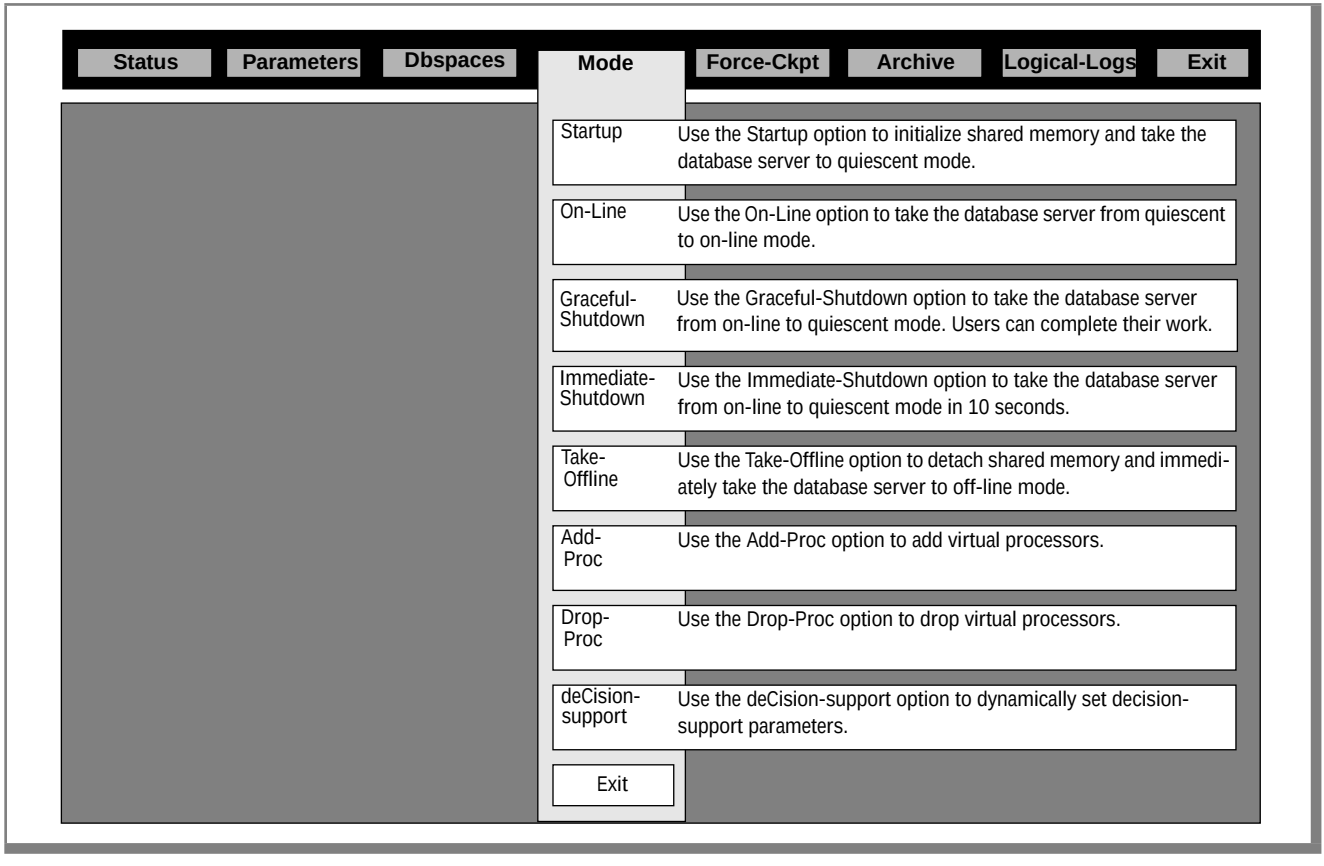


Figure 32-5
Force-Ckpt Option

Figure 32-6
Archive Menu

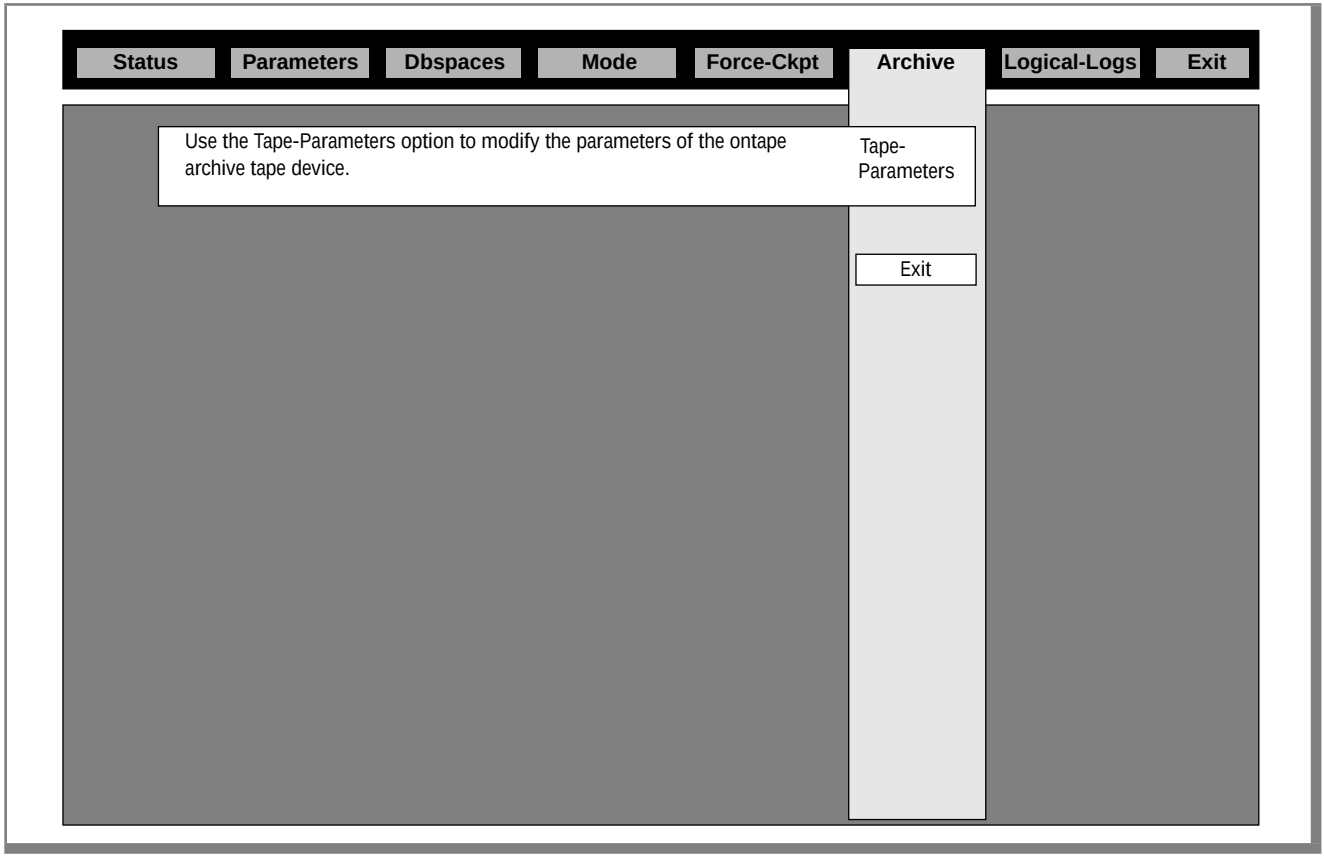
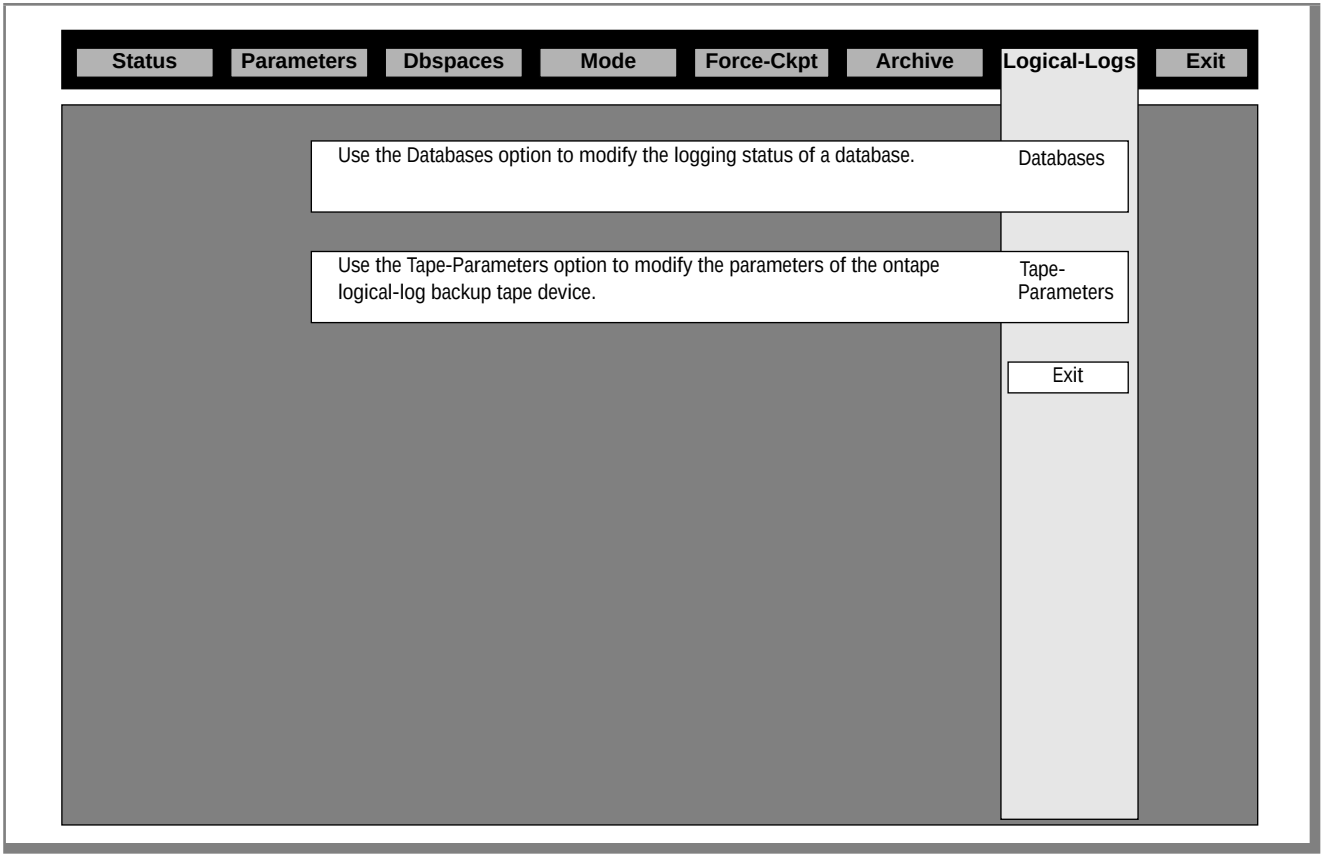


Figure 32-7
Logical-Logs Menu



Setting Configuration Parameters with ON-Monitor

Each ON-Monitor parameters screen is represented here by a pair of examples (Figure 32-8 through Figure 32-18). The first example in each pair shows the ON-Monitor screen. The second example in each pair shows which parameters from the ONCONFIG configuration file correspond to which fields in the parameters screen.

```
 INITIALIZATION: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                              DISK PARAMETERS
Page Size     [    2] Kbytes                                  Mirror [N]

Tape Dev.     [/dev/tapedev                                             ]
Block Size    [        16] Kbytes           Total Tape Size [     10240] Kbytes
Log Tape Dev. [/dev/tapedev                                             ]
Block Size    [        16] Kbytes           Total Tape Size [     10240] Kbytes
Stage Blob    [                  ]

Root Name     [rootdbs           ]               Root Size [     20000] Kbytes
Primary Path  [/dev/cynsrv_root                                         ]
                                                Root Offset [         0] Kbytes
Mirror Path   [                                                         ]
                                              Mirror Offset [         0] Kbytes
Phy. Log Size [      1000] Kbytes               Log. Log Size [       500] Kbytes
                                           Number of Logical Logs [  6  ]
```

Figure 32-8
INITIALIZATION Screen

```
 INITIALIZATION: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                              DISK PARAMETERS
Page Size     {not in ONCONFIG}                          Mirror {MIRROR}

Tape Dev.     {TAPEDEV}
Block Size    {TAPEBLK}                    Total     Tape Size {TAPESIZE}
Log Tape Dev. {LTAPEDEV}
Block Size    {LTAPEBLK}                   Total    Tape Size {LTAPESIZE}
Stage Blob    {STAGEBLOB}

Root Name     {ROOTNAME}                              Root Size {ROOTSIZE}
Primary Path  {ROOTPATH}
                                                   Root Offset {ROOTOFFSET}
Mirror Path   {MIRRORPATH}
                                                 Mirror Offset {MIRROROFFSET}
Phy. Log Size {PHYSFILE}                         Log. Log Size {LOGSIZE}
                                              Number of Logical Logs {LOGFILES}
```

Figure 32-9
INITIALIZATION Screen Showing Parameter Names

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                     SHARED MEMORY PARAMETERS
Server Number                  [   0]       Server Name [a_server          ]
Server Aliases [                                                          ]
Dbspace Temp   [                                                          ]
Deadlock Timeout       (Secs) [ 60]    Number of Page Cleaners    [   1]
Forced Residency               [N]    Stack Size (Kbytes)        [  32]
Non Res. SegSize      (Kbytes)  [8000]    Optical Cache Size (Kbytes)[     0]

Physical Log Buffer Size [   32] Kbytes  Dbspace Down Option         [0]
Logical Log Buffer Size  [   32] Kbytes  Preserve Log for Log Backup [N]
Max # of Logical Logs          [    6]      Transaction Timeout        [ 300]
Max # of Locks       [        2000]      Long TX HWM               [  50]
Max # of Buffers      [        200]      Long TX HWM Exclusive     [  60]
                                        Index Page Fill Factor    [  90]
                                        Add SegSize (Kbytes) [     8192]
                                        Total Memory(Kbytes) [        0]

Resident Shared memory size   [   864] Kbytes   Page Size [    2] Kbytes

Enter a unique value to be associated with this version of Informix Dynamic Server
```

Figure 32-10
Shared Memory Screen

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                     SHARED MEMORY PARAMETERS
Server Number       {SERVERNUM}              Server Name          {DBSERVERNAME}
Server Aliases      {DBSERVERALIASES}
Dbspace Temp        {DBSPACETEMP}
Deadlock Timeout    {DEADLOCK_TIMEOUT}       Number of Page Cleaners  {CLEANERS}
Forced Residency    {RESIDENCY}              Stack Size (Kbytes)     {STACKSIZE}
Non Res. SegSize    {SHMVIRTSIZE}         Optical Cache Size  {OPCACHEMAX}

Physical Log Buffer Size  {PHYSBUFF}    Dbspace Down Option   {ONDBSPDOWN}
Logical Log Buffer Size{LOGBUFF} Preserve Log for Log Backup{LBU PRESERVE}
Max # of Logical Logs     {LOGSMAX}       Transaction Timeout        {TXTIMEOUT}
Max # of Locks     {LOCKS}             Long TX HWM                {LTXHWM}
Max # of Buffers      {BUFFERS}        Long TX HWM Exclusive     {LTXEHWM}
                                       Index Page Fill Factor {FILLFACTOR}
                                       Add SegSize                {SHMADD}
                                       Total Memory             {SHMTOTAL}

Resident Shared memory size    [     ] Kbyte   Page Size  {not in ONCONFIG}
Enter a unique value to be associated with this version of Informix Dynamic Server
```

Figure 32-11
SHARED MEMORY Screen Showing Parameter Names

```
PERFORMANCE: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                    PERFORMANCE TUNING PARAMETERS

 Multiprocessor Machine      [N]     LRU Max Dirty             [ 60]
   Num Procs to Affinity   [  0]     LRU Min Dirty             [ 50]
   Proc num to start with  [  0]     Checkpoint Interval      [ 300]
                                     Num of Read Ahead Pages   [  4]
 CPU VPs                   [  1]     Read Ahead Threshold      [  2]
 AIO VPs                   [  2]
 Single CPU VP               [N]     NETTYPE settings:
 Use OS Time                 [N]       Protocol Threads Users VP-class
 Disable Priority Aging      [N]       [      ] [   ]   [   ] [   ]
 Off-Line Recovery Threads [ 10]       [      ] [   ]   [   ] [   ]
 On-Line Recovery Threads  [  1]       [      ] [   ]   [   ] [   ]
 Num of LRUS queues        [  8]       [      ] [   ]   [   ] [   ]




Are you running on a multiprocessor machine?
```

Figure 32-12
PERFORMANCE Screen

```
PERFORMANCE: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                    PERFORMANCE TUNING PARAMETERS

              {MULTIPROCESSOR}                    {LRU_MAX_DIRTY}
                  {AFF_NPROCS}                    {LRU_MIN_DIRTY}
                  {AFF_SPROC}                         {CKPTINTVL}
                                                       {RA_PAGES}
                   {NUMCPUVPS}                    {RA_THRESHOLD}
                   {NUMAIOVPS}
                {SINGLE_CPU_VP}        NETTYPE settings:
                  {USE_OS_TIME}        (NETTYPE values are assembled
                         {NOAGE}        from the responses in these
           {OFF_RECVRY_THREADS}         blanks)
            {ON_RECVRY_THREADS}
                        {LRUS}




Are you running on a multiprocessor machine?
```

Figure 32-13
PERFORMANCE Screen Showing Parameter Names

```
DATA REPLICATION: Make desired changes and press ESC to record changes.
  Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                        DATA REPLICATION PARAMETERS

 Interval       [  30]
 Timeout        [  30]
 Auto           [0]
 Lost & Found   [/usr/informix/etc/dr.lostfound                          ]
```

Figure 32-14
DATA REPLICATION Screen

```
DATA REPLICATION: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                        DATA REPLICATION PARAMETERS

 Interval       {DRINTERVAL}
 Timeout        {DRTIMEOUT}
 Auto           {DRAUTO}
 Lost & Found   {DRLOSTFOUND}
```

Figure 32-15
DATA REPLICATION Screen Showing Parameter Names

```
DIAGNOSTICS: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                      DIAGNOSTIC PARAMETERS

Message Log   [/usr/informix/server.log                                  ]
Console Msgs. [/dev/console                                              ]
Alarm Program [                                                          ]

Dump Shared Memory       [Y]
Dump Gcore               [N]
Dump Core                [N]
Dump Count               [   1]
Dump Directory           [/tmp                           ]
```

Figure 32-16
DIAGNOSTICS Screen

```
DIAGNOSTICS: Make desired changes and press ESC to record changes.
 Press Interrupt to abort changes.  Press F2 or CTRL-F for field-level help.
                      DIAGNOSTIC PARAMETERS

Message Log              {MSGPATH}
Console Msgs.            {CONSOLE}
Alarm Program            {ALARMPROGRAM}

Dump Shared Memory       {DUMPSHMEM}
Dump Gcore               {DUMPGCORE}
Dump Core                {DUMPCORE}
Dump Count               {DUMPCNT}
Dump Directory           {DUMPDIR}
```

Figure 32-17
DIAGNOSTICS Screen Showing Parameter Names

```
PDQ: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                    PARALLEL DATABASE QUERIES PARAMETERS


Maximum Priority                  [100]
Decision Support Queries          [         ]
Decision Support Memory (Kbytes)  [         ]
Maximum Decision Support Scans    [        3]
Dataskip                          [         ]
Optimizer Hint                    [2]
```

Figure 32-18
PDQ Screen

```
PDQ: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                    PARALLEL DATABASE QUERIES PARAMETERS


Maximum Priority                  {MAX_PDQPRIORITY}
Decision Support Queries          {DS_MAX_QUERIES}
Decision Support Memory (Kbytes)  {DS_TOTAL_MEMORY}
Maximum Decision Support Scans    {DS_MAX_SCANS}
Dataskip                          {DATASKIP}
Optimizer Hint                    {OPTCOMPIND}
```

Figure 32-19
PDQ Screen Showing Parameter Names

Chapter

33

Configuration Parameters

ONCONFIG File Conventions 33-5

ONCONFIG Parameters 33-6

ADTERR, ADTMODE, ADTPATH, and ADTSIZE 33-8

AFF_NPROCS 33-9

AFF_SPROC 33-10

ALARMPROGRAM 33-11

BAR_ACT_LOG, BAR_BSALIB_PATH, BAR_MAX_BACKUP, BAR_NB_XPORT_COUNT, BAR_RETRY, and BAR_XFER_BUF_SIZE 33-11

BUFFERS 33-12

CKPTINTVL 33-13

CLEANERS 33-14

CONSOLE 33-15

DATASKIP 33-16

DBSERVERALIASES 33-17

DBSERVERNAME 33-18

DBSPACETEMP 33-19

DEADLOCK_TIMEOUT 33-21

DRAUTO 33-22

DRINTERVAL 33-23

DRLOSTFOUND 33-24
DRTIMEOUT 33-25
DS_MAX_QUERIES 33-26
DS_MAX_SCANS 33-27
DS_TOTAL_MEMORY 33-28
DUMPCNT 33-32
DUMPCORE 33-33
DUMPDIR 33-34
DUMPGCORE 33-34
DUMPSHMEM 33-35
FILLFACTOR 33-36
HETERO_COMMIT 33-36
LBU_PRESERVE 33-37
LOCKS 33-38
LOG_BACKUP_MODE 33-39
LOGBUFF 33-40
LOGFILES 33-41
LOGSIZE 33-42
LOGSMAX. 33-43
LRUS. 33-44
LRU_MAX_DIRTY 33-45
LRU_MIN_DIRTY 33-45
LTAPEBLK. 33-46
LTAPEDEV 33-47
LTAPESIZE 33-48

LTXEHWM 33-49
LTXHWM 33-50
MAX_PDQPRIORITY 33-51
MIRROR 33-52
MIRROROFFSET 33-53
MIRRORPATH 33-53
MSGPATH 33-54
MULTIPROCESSOR 33-55
NETTYPE. 33-56
NOAGE 33-60
NUMAIOVPS 33-61
NUMCPUVPS 33-62
OFF_RECVRY_THREADS 33-63
ON_RECVRY_THREADS 33-64
ONDBSPACEDOWN. 33-65
OPCACHEMAX 33-66
OPTCOMPIND. 33-67
PHYSBUFF 33-68
PHYSDBS. 33-69
PHYSFILE 33-70
RA_PAGES 33-71
RA_THRESHOLD. 33-72
RESIDENT 33-73
RESTARTABLE_RESTORE. 33-74
ROOTNAME 33-75

ROOTOFFSET 33-75
ROOTPATH 33-76
ROOTSIZE 33-77
SERVERNUM 33-78
SHMADD 33-79
SHMBASE 33-80
SHMTOTAL 33-81
SHMVIRTSIZE 33-82
SINGLE_CPU_VP 33-83
STACKSIZE 33-84
STAGEBLOB 33-85
TAPEBLK 33-86
TAPEDEV 33-87
TAPESIZE 33-89
TXTIMEOUT 33-90
USEOSTIME 33-91

This chapter describes the ONCONFIG file conventions and the ONCONFIG parameters.

ONCONFIG File Conventions

The **ONCONFIG** environment variable specifies the file that contains the configuration parameters. This manual often refers to this file as "the ONCONFIG file." The database server uses the ONCONFIG file during initialization.

The database server provides **onconfig.std**, which is a template for a configuration file. It contains initial values for many of the ONCONFIG parameters. The following table lists the locations of the ONCONFIG and **onconfig.std** files for each platform.

Platform	ONCONFIG File	Template File
UNIX	**$INFORMIXDIR/etc/$ONCONFIG**	**$INFORMIXDIR/etc/onconfig.std**
Windows NT	**%INFORMIXDIR%\etc\%ONCONFIG%**	**%INFORMIXDIR%\etc\onconfig.std**

To prepare the ONCONFIG file

1. Copy **onconfig.std**.
2. Modify the *copy* of **onconfig.std**.
3. Set **ONCONFIG** to the name of the copy of **onconfig.std**.

   If you do not set ONCONFIG, the default filename is **onconfig**.

Warning: *Do not modify* ***onconfig.std****.*

For more information about the **ONCONFIG** environment variable, see the *Informix Guide to SQL: Reference*. For more information about the **onconfig.std**, **onconfig**, and ONCONFIG files, see Appendix A.

ONCONFIG Parameters

In the ONCONFIG file, each parameter is on a separate line. The file can also contain blank lines and comment lines that start with a # symbol. The following line shows the syntax for a parameter line:

```
PARAMETER_NAME    parameter_value    #comment
```

Parameters and their values in the ONCONFIG file are case sensitive. The parameter names consist entirely of uppercase letters. If the range of values for a parameter is in uppercase, use uppercase for the value in the ONCONFIG file. For example, one of the possible values for NETTYPE is `CPU`.

Put white space (tabs, spaces, or both) between the parameter name, parameter value, and optional comment. Do not use any tabs or spaces within a parameter value.

Parameter Attributes

The description for each parameter can include any of the following attributes:

onconfig.std* *value	The value that is in the **onconfig.std** file
if not present	The value that the database server supplies if the parameter is missing from your ONCONFIG file
units	The units in which the parameter is expressed
separators	The separators to use when the value has several parts. Do *not* use white space within a parameter value.
range of values	The range of possible values

takes effect	The time at which a change to the value of the parameter affects the operation of the database server
UNIX utilities	The UNIX utilities that you can use to change the value of the parameter For ON-Monitor, this section lists the menu choices and item name that display the parameter. This section can also list command-line utilities.
Windows NT utilities	The command-line utility that you can use to change the value of the parameter
refer to	Cross references to related information

UNIX

Changing a Parameter Value on UNIX

Use one of the utilities in the following table to change the value of a configuration parameter.

Utility	Description
ON-Monitor	In the attributes section in each parameter description, the *UNIX utilities* section lists the menu choices and item name that display the parameter in ON-Monitor. In ON-Monitor, some of the responses are `Y/N` (yes/no). When those responses are recorded in the ONCONFIG file, `Y` becomes `1` and `N` becomes `0`.
A command-line utility	In the attributes section in the descriptions for most parameters, the *UNIX utilities* section lists one or more command-line utilities that you can use to change a parameter value.

WIN NT

Changing a Parameter Value on Windows NT

Use one of the tools in the following table to change the value of a configuration parameter.

Tools	Description
A text editor	You can use a text editor to modify the ONCONFIG file.
A command-line utility	In the attributes section in the descriptions for most parameters, the *Windows NT utilities* section lists one or more command-line utilities that you can use to change a parameter value.

UNIX

ADTERR, ADTMODE, ADTPATH, and ADTSIZE

The ADTERR, ADTMODE, ADTPATH, and ADTSIZE parameters are auditing configuration parameters. For information on these parameters, see your *Trusted Facility Manual*.

AFF_NPROCS

onconfig.std *value*	0
units	Number of CPUs
range of values	0 through (number of CPUs in the computer)
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Num Procs to Affinity
refer to	"CPU Virtual Processors" on page 9-20 "AFF_SPROC" on page 33-10

On multiprocessor computers that support *processor affinity,* AFF_NPROCS specifies the number of CPUs to which the database server can bind CPU virtual processors. Binding a CPU virtual processor to a CPU causes the virtual processor to run exclusively on that CPU. The database server assigns CPU virtual processors to CPUs in serial fashion, starting with the processor number specified by AFF_SPROC.

AFF_SPROC

onconfig.std *value*	0
units	CPU number
range of values	0 through (AFF_NPROCS - NUMCPUVPS + 1)
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Proc num (to start with)
refer to	"CPU Virtual Processors" on page 9-20

On multiprocessor computers that support *processor affinity*, AFF_SPROC specifies the CPU, starting with 0, on which the database server starts binding CPU virtual processors to CPUs. The AFF_NPROCS parameter specifies the number of CPUs that the database server will use. The NUMCPUVPS parameter specifies the number of CPU virtual processors that the database server will start, and the AFF_SPROC parameter specifies the CPU, of the number specified by AFF_NPROCS, on which the database server starts the first virtual processor. For example, if your platform has eight CPUs, and you set AFF_NPROCS to 3 and AFF_SPROC to 5, the database server binds CPU virtual processors to CPUs 6, 7, and 8.

ALARMPROGRAM

onconfig.std *value*	On UNIX: **/usr/informix/log_full.sh** On Windows NT: None
range of values	Pathname
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics
refer to	"Event Alarm" on page 29-8

Set ALARMPROGRAM to the full pathname of an executable file that you write and that the database server executes when noteworthy events occur. Noteworthy events include database, table, index, or blob failure; chunk or dbspace taken off-line; internal subsystem failure; initialization failure; and detection of long transaction.

BAR_ACT_LOG, BAR_BSALIB_PATH, BAR_MAX_BACKUP, BAR_NB_XPORT_COUNT, BAR_RETRY, and BAR_XFER_BUF_SIZE

The BAR_ACT_LOG, BAR_BSALIB_PATH, BAR_MAX_BACKUP, BAR_NB_XPORT_COUNT, BAR_RETRY, and BAR_XFER_BUF_SIZE parameters apply to the ON-Bar backup and restore system. For information on these parameters, see your *Backup and Restore Guide*.

BUFFERS

onconfig.std *value*	200
units	Number of bytes
range of values	For UNIX: 100 through 768 kilobytes (768 * 1024) For Windows NT: 100 through 512 kilobytes (512 * 1024)
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared_Memory, Max # of Buffers
refer to	"Shared-Memory Buffer Pool" on page 11-19 "RA_PAGES" on page 33-72 "RA_THRESHOLD" on page 33-73 *Performance Guide*

BUFFERS specifies the maximum number of shared-memory buffers that the database server user threads have available for disk I/O on behalf of client applications. Therefore, 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.

In general, buffer space should range from 20 to 25 percent of physical memory. Informix recommends that you calculate all other shared-memory parameters after you set buffer space (BUFFERS * system page size) to 20 percent of physical memory.

If you also want to perform read-ahead, increase the value of BUFFERS further. Once you have configured all other shared-memory parameters, if you find that you can afford to increase the size of shared memory, increase the value of BUFFERS until buffer space reaches the recommended 25 percent maximum.

System Page Size

The system page size is platform dependent. You can use the commands in the following table to display the system page size.

Command	Description
oncheck -pr	Checks the root-dbspace reserved pages and displays the system page size in the first section of its output
onstat -b	Displays the system page size, given as buffer size

UNIX

On UNIX, you can also use ON-Monitor to get the system page size under the Parameters:Shared-Memory option, which does not require the database server to be running, and under the Parameters:Initialize option. ♦

CKPTINTVL

onconfig.std *value*	300
units	Seconds
range of values	Any value greater than or equal to 0
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Checkpoint Interval
refer to	"Checkpoints" on page 11-57 *Performance Guide*

CKPTINTVL specifies the frequency, expressed in seconds, at which the database server will check to determine whether a checkpoint is needed. When a checkpoint occurs, pages in the shared-memory buffer pool are written to disk.

If you set CKPTINTVL to an interval that is too short, the system spends too much time performing checkpoints, and the performance of other work suffers. If you set CKPTINTVL to an interval that is too long, fast recovery might take too long.

In practice, 30 seconds is the smallest interval that the database server checks. If you specify a checkpoint interval of 0, the database server will not check if the checkpoint interval has elapsed. However, the database server will still perform checkpoints. Other conditions, such as the physical log becoming 75 percent full, also cause the database server to perform checkpoints.

CLEANERS

onconfig.std *value*	1
units	Number of page-cleaner threads
range of values	1 through 128
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Number of Page Cleaners
refer to	"How the Database Server Flushes Data to Disk" on page 11-46

CLEANERS specifies the number of page-cleaner threads available during the database server operation. By default, the database server always runs one page-cleaner thread. A general guideline is one page cleaner per disk drive. The value specified has no effect on the size of shared memory.

CONSOLE

onconfig.std *value*	On UNIX: **/dev/console** On Windows NT: **\etc\console.log**
range of values	Pathname
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Console Msgs.
refer to	"What Is the Console?" on page 29-12

CONSOLE specifies the pathname destination for console messages.

DATASKIP

syntax	DATASKIP *state* or DATASKIP *state dbspace1 dbspace2 ...* The *state* entry is required. If *state* is ON, then at least one *dbspace* entry is required.
onconfig.std *value*	None
if not present	OFF
separators	Space
range of values	ALL = Skip all unavailable fragments. OFF = Turn off dataskip. ON = Skip some unavailable fragments.
UNIX utilities	ON-Monitor: Parameters menu, pdQ screen **onspaces -f**; see page 35-61
Windows NT utilities	**onspaces -f**; see page 35-61
refer to	"Specify DATASKIP Parameter" on page 35-61 and your *Performance Guide*

DATASKIP lets you avoid points of media failure. This capability can result in higher availability for your data. To instruct the database server to skip some or all unavailable fragments, set this parameter.

Whenever the database server skips over a dbspace during query processing, a warning is returned. The previously reserved SQLCA warning flag **sqlwarn.sqlwarn7** is set to W for INFORMIX-ESQL/C.

An application can use the SQL statement SET DATASKIP to override the DATASKIP value set by the ONCONFIG parameter or by **onspaces**. If the application then executes the SQL statement SET DATASKIP DEFAULT, the DATASKIP value for that session returns to whatever value is currently set for the database server.

DBSERVERALIASES

onconfig.std *value*	None
if not present	None
separators	Comma
range of values	Up to 18 lowercase characters
takes effect	When shared memory is initialized. In addition, the file or registry of each database server might need to be updated.
UNIX utilities	Parameters, Shared-Memory, Server Aliases
refer to	"ONCONFIG Parameters for Connectivity" on page 4-50 "Using Multiple Connection Types" on page 4-57

DBSERVERALIASES specifies a list of alternative dbservernames. If the database server supports more than one communication protocol (for example, both an IPC mechanism and the TCP network protocol), you must describe each valid connection to the database server with an entry in the **sqlhosts** file or registry. DBSERVERALIASES lets you assign multiple aliases to a database server, so that each entry in the **sqlhosts** file or registry can have a unique name.

For each alternate name listed in DBSERVERALIASES, the database server starts an additional listener thread. If you have many client applications connecting to the database server, you can distribute the connection requests between several listener threads and reduce connection time. To take advantage of the alternate connections, instruct some of your client applications to use a CONNECT TO *dbserveralias* statement instead of CONNECT TO *dbservername.*

DBSERVERNAME

onconfig.std *value*	None
if not present	On UNIX: *hostname* On Windows NT: ol_*hostname* (*hostname* is the name of the host computer.)
range of values	Up to 18 lowercase characters. The first character must be a letter. DBSERVERNAME can include any printable character except the following characters: ■ Uppercase characters ■ A field delimiter (space or tab) ■ A newline character ■ A comment character
takes effect	When shared memory is initialized. The **sqlhosts** file or registry of each database server that communicates with this database server might need to be updated. In addition, the **INFORMIXSERVER** environment variable for all users might need to be changed.
UNIX utilities	Parameters, Shared-Memory, Server Name
refer to	"The DBSERVERNAME Configuration Parameter" on page 4-51

DBSERVERNAME specifies a unique name associated with this specific occurrence of the database server. The value of DBSERVERNAME is called the *dbservername*. Each dbservername is associated with a communication protocol in the **sqlhosts** file or registry. If the database server uses multiple communication protocols, additional values for dbservername must be defined with the DBSERVERALIASES configuration parameter.

Client applications use the dbservername in the **INFORMIXSERVER** environment variable and in SQL statements such as CONNECT and DATABASE, which establish a connection to a database server.

Informix recommends that you use DBSERVERNAME to assign a name instead of using the default name because of possible conflict with other database servers on the same host computer

DBSPACETEMP

onconfig.std *value*	None
if not present	ROOTNAME
separators	Comma or colon (no white space)
range of values	A list of dbspaces. The length of the list cannot exceed 254 characters.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Dbspace Temp
refer to	"What Is a Temporary Table?" on page 13-24 "Where Are Temporary Tables Stored?" on page 13-26 *Informix Guide to SQL: Reference*

DBSPACETEMP specifies a list of dbspaces that the database server uses to manage globally the storage of temporary tables. Use of this parameter enables the database server to spread out I/O for temporary tables efficiently across multiple disks. The database server also uses temporary dbspaces during backups to store the before images of data that is overwritten while the backup is occurring.

The list of dbspaces can contain standard dbspaces, temporary dbspaces, or both. Use a colon or comma to separate the dbspaces in your list. If both standard and temporary dbspaces are listed in the DBSPACETEMP configuration parameter or environment variable, the following rules apply:

- Sort, backup, implicit, and nonlogging explicit temporary tables are created in temporary dbspaces if adequate space exists.
- Explicit temporary tables created without the WITH NO LOG option are created in standard (rather than temporary) dbspaces.

You can use the **PSORT_DBTEMP** environment variable to specify a list of directories to be used for the temporary storage of implicit sort files. For the order of precedence that the database server uses when it creates implicit sort files, see your *Performance Guide*.

If a client application needs to specify an alternative list of dbspaces to use for its temporary-table locations, the client can enumerate them by using the **DBSPACETEMP** environment variable.

Hash Joins Overflow and DBSPACETEMP

If you do not set the **DBSPACETEMP** environment variable or DBSPACETEMP configuration parameter, and the optimizer chooses to perform a join or group-by operations using a hash join, the database server directs any overflow that results from the join to the **tmp** directory.

When Changes to DBSPACETEMP Take Effect

When you create a temporary dbspace with ON-Monitor (UNIX only) or **onspaces**, the database server does not use the newly created temporary dbspace until you perform the following steps:

1. Add the name of a new temporary dbspace to your list of temporary dbspaces in the DBSPACETEMP configuration parameter, the DBSPACETEMP environment variable, or both.
2. Reinitialize the database server.

DEADLOCK_TIMEOUT

onconfig.std *value*	60
units	Seconds
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Deadlock Timeout
refer to	"Configuration Parameters Used in Two-Phase Commits" on page 30-37

DEADLOCK_TIMEOUT specifies the maximum number of seconds that a database server thread can wait to acquire a lock.

This parameter is used only for distributed queries that involve a remote database server. Nondistributed queries do not use this parameter.

DRAUTO

onconfig.std *value*	0
range of values	0 or OFF = Do not automatically switch. 1 or RETAIN_TYPE = Automatically switch secondary to standard on a high-availability data-replication failure. Switch back to secondary when restarting high-availability data replication. 2 or REVERSE_TYPE = Automatically switch secondary to standard on a high-availability data-replication failure. Switch to primary (and switch original primary to secondary) when restarting high-availability data replication.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, data-Replication, Auto
refer to	"What Is Automatic Switchover?" on page 25-19

DRAUTO determines how a secondary database server reacts to a high-availability data-replication failure. This parameter should have the same value on both high-availability data-replication servers.

If DRAUTO is set to OFF, the secondary database server remains a secondary database server in read-only mode when a high-availability data-replication failure occurs.

If DRAUTO is set to either RETAIN_TYPE or REVERSE_TYPE, the secondary database server switches to type standard automatically when a high-availability data-replication failure is detected. If DRAUTO is set to RETAIN_TYPE, the original secondary database server switches back to type secondary when the high-availability data-replication connection is restored. If DRAUTO is set to REVERSE_TYPE, the original secondary database server switches to type primary when the high-availability data-replication connection is restored, and the original primary switches to type secondary.

Use this parameter carefully. A network failure (that is, when the primary database server does not really fail, but the secondary database server perceives network slowness as a high-availability data-replication failure) can cause the two database servers to become out of synch.

DRINTERVAL

onconfig.std *value*	30
units	Seconds
range of values	-1, 0, and positive integer values
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, data-Replication, Interval
refer to	"When Are Log Records Sent?" on page 25-10

DRINTERVAL specifies the maximum time interval in seconds between flushing of the high-availability data-replication buffer. To update synchronously, set the parameter to -1.

DRLOSTFOUND

onconfig.std	On UNIX: **/usr/etc/dr.lostfound**
value	On Windows NT: **\tmp**
range of values	Pathname
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, data-Replication, Lost & Found
refer to	"Lost-and-Found Transactions" on page 25-11

DRLOSTFOUND specifies the pathname 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. The file is created with a time stamp appended to the filename, so that the database server does not overwrite another lost-and-found file if one already exists.

This parameter is not applicable if updating between the primary and secondary database servers occurs synchronously (that is, if DRINTERVAL is set to `-1`).

DRTIMEOUT

onconfig.std *value*	30
units	Seconds
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, data-Replication, Timeout
refer to	"How Are High-Availability Data-Replication Failures Detected?" on page 25-16

DRTIMEOUT applies only to high-availability data-replication pairs. This value specifies the length of time, in seconds, that a database server in a high-availability data-replication pair waits for a transfer acknowledgment from the other database server in the pair. Use the following formula to calculate DRTIMEOUT:

```
DRTIMEOUT = wait_time / 4
```

In this formula, *wait_time* is the length of time, in seconds, that a database server in a high-availability data-replication pair must wait before it assumes that a high-availability data-replication failure occurred.

For example, suppose you determine that the *wait_time* for your system is 160 seconds. Use the preceding formula to set DRTIMEOUT as follows:

```
DRTIMEOUT = 160 seconds / 4 = 40 seconds
```

DS_MAX_QUERIES

onconfig.std *value*	On UNIX: None On Windows NT: 32
if not present	**NUMCPUVPS** * 2 * 128
units	Number of queries
range of values	1 through (8 * 1024 * 1024)
UNIX utilities	ON-Monitor: Parameters menu, pdQ screen **onmode -Q**; see page 35-39.
Windows NT utilities	**onmode -Q**; see page 35-39.
refer to	*Performance Guide*

DS_MAX_QUERIES is the maximum number of queries that can run concurrently. The memory grant manager (MGM) reserves memory for a query based on the following formula:

```
memory_reserved = (DS_TOTAL_MEMORY / DS_MAX_QUERIES) *
                  DS_MAX_QUERIES * (PDQPRIORITY / 100) *
                  (MAX_PDQPRIORITY / 100)
```

DS_MAX_SCANS

onconfig.std *value*	1,048,576 (1024 * 1024)
units	Number of PDQ scan threads
range of values	10 through (1024 * 1024)
UNIX utilities	ON-Monitor: Parameters menu, pdQ screen **onmode -S**; see page 35-39.
Windows NT utilities	**onmode -S**; see page 35-39.
refer to	*Performance Guide*

DS_MAX_SCANS limits the number of PDQ scan threads that the database server can execute concurrently. When a user issues a query, the database server apportions some number of scan threads, depending on the following values:

- The value of PDQ priority (set by the environment variable **PDQPRIORITY** or the SQL statement SET PDQPRIORITY)
- The ceiling that you set with DS_MAX_SCANS
- The factor that you set with MAX_PDQPRIORITY
- The number of fragments in the table to scan (*nfrags* in the formula)

The memory grant manager (MGM) tries to reserve scan threads for a query according to the following formula:

```
reserved_threads = min (nfrags, (DS_MAX_SCANS *
                        PDQPRIORITY / 100 *
                        MAX_PDQPRIORITY / 100) )
```

If the DS_MAX_SCANS part of the formula is greater than or equal to the number of fragments in the table to scan, then the query is held in the ready queue until as many scan threads are available as there are table fragments. Once underway, the query executes quickly because threads are scanning fragments in parallel.

For example, if *nfrags* equals 24, DS_MAX_SCANS equals 90, **PDQPRIORITY** equals 50, and MAX_PDQPRIORITY equals 60, the query will not begin execution until *nfrags* scan threads are available. Scanning will take place in parallel.

If you cause the DS_MAX_SCANS formula to fall below the number of fragments, the query might begin execution sooner, but the query will take longer to execute because some threads will scan fragments serially.

If you reduce DS_MAX_SCANS to 40 in the previous example, the query needs fewer resources (12 scan threads) to begin execution, but each thread will need to scan two fragments serially. Execution will take longer.

DS_TOTAL_MEMORY

onconfig.std *value*	On UNIX: None On Windows NT: 4,096
if not present	If SHMTOTAL=0 and DS_MAX_QUERIES is set: DS_MAX_QUERIES * 128. If SHMTOTAL=0 and DS_MAX_QUERIES is not set: NUMCPUVPS * 2 * 128. If SHMTOTAL>0: SHMTOTAL - (all non-DS memory). Non-DS memory includes the resident memory segment, memory reserved for buffer cache, and memory reserved for user connections in various protocols, which is specified by the third argument of NETTYPE (default is 50 connections).
units	Kilobytes
range of values	If DS_MAX_QUERY is set: DS_MAX_QUERY * 128 through *maxmem* If DS_MAX_QUERY is not set: NUMCPUVPS * 2 * 128 through *maxmem*

UNIX utilities	ON-Monitor: Parameters menu, pdQ screen **onmode -M**; see page 35-39.
Windows NT utilities	**onmode -M**; see page 35-39.
refer to	*Performance Guide* "Derive a Minimum for Decision-Support Memory" on page 33-30

DS_TOTAL_MEMORY specifies the amount of memory available for PDQ queries. It should be smaller than the computer physical memory, minus fixed overhead such as operating-system size and buffer-pool size.

Do not confuse DS_TOTAL_MEMORY with the configuration parameter SHMTOTAL. For OLTP applications, set DS_TOTAL_MEMORY to between 20 and 50 percent of the value of SHMTOTAL in kilobytes. For applications that involve large decision-support queries, increase the value of DS_TOTAL_MEMORY to between 50 and 80 percent of SHMTOTAL. If your database server instance is used exclusively for decision-support queries, set this parameter to 90 percent of SHMTOTAL.

Algorithm for DS_TOTAL_MEMORY

The database server derives a value for DS_TOTAL_MEMORY when you do not set DS_TOTAL_MEMORY or if you set it to an inappropriate value. Whenever the database server changes the value that you assigned to DS_TOTAL_MEMORY, it notifies you by sending the following message to your console:

```
DS_TOTAL_MEMORY recalculated and changed from old_value Kb to new_value Kb
```

The algorithm that the database server uses to derive the new value for DS_TOTAL_MEMORY is documented in the following sections. When you receive the preceding message, you can use the algorithm to investigate what values the database server considers inappropriate and take corrective action based on your investigation.

Derive a Minimum for Decision-Support Memory

In the first part of the algorithm, the database server establishes a minimum for decision- support memory. When you assign a value to the configuration parameter DS_MAX_QUERIES, the database server sets the minimum amount of decision-support memory according to the following formula:

```
min_ds_total_memory = DS_MAX_QUERY * 128Kb
```

When you do not assign a value to DS_MAX_QUERIES, the database server instead uses the following formula based on the value of NUMCPUVPS:

```
min_ds_total_memory = (number_of_CPU_VPS) * 2 * 128Kb
```

Derive a Working Value for Decision-Support Memory

In the second part of the algorithm, the database server establishes a working value for the amount of decision-support memory. The database server verifies this amount in the third and final part of the algorithm.

When DS_TOTAL_MEMORY Is Set

The database server first checks if SHMTOTAL is set. When SHMTOTAL is set, the database server uses the following formula to calculate DS_TOTAL_MEMORY:

```
IF DS_TOTAL_MEMORY <= SHMTOTAL - nondecision_support_memory THEN
   decision_support_memory = DS_TOTAL_MEMORY
ELSE
    decision_support_memory = SHMTOTAL - nondecision_support_memory
```

This algorithm effectively prevents you from setting DS_TOTAL_MEMORY to values that the database server cannot possibly allocate to decision-support memory.

When SHMTOTAL is not set, the database server sets decision-support memory equal to the value that you specified in DS_TOTAL_MEMORY.

When DS_TOTAL_MEMORY Is Not Set

When you do not set DS_TOTAL_MEMORY, the database server proceeds as follows. First, the database server checks if you set SHMTOTAL. When SHMTOTAL is set, the database server uses the following formula to calculate the amount of decision-support memory:

```
decision_support_memory = SHMTOTAL - nondecision_support_memory
```

When the database server finds that you did not set SHMTOTAL, it sets decision-support memory as shown in the following example:

```
decision_support_memory = min_ds_total_memory
```

Check Derived Value for Decision-Support Memory

The final part of the algorithm verifies that the amount of shared memory is greater than *min_ds_total_memory* and less than the maximum possible memory space for your computer. When the database server finds that the derived value for decision-support memory is less than *min_ds_total_memory*, it sets decision-support memory equal to *min_ds_total_memory*.

When the database server finds that the derived value for decision-support memory is greater than the maximum possible memory space for your computer, it sets decision-support memory equal to the maximum possible memory space.

Inform User When Derived Value Is Different from User Value

When, at any point during the processing of this algorithm, the database server changes the value that you set for DS_TOTAL_MEMORY, it sends a message to your console in the following format:

```
DS_TOTAL_MEMORY recalculated and changed from old_value Kb to new_value Kb
```

In the message, *old_value* represents the value that you assigned to DS_TOTAL_MEMORY in your configuration file, and *new_value* represents the value that the database server derived.

UNIX

DUMPCNT

onconfig.std *value*	1
if not present	1
units	Number of assertion failures
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Dump Count
refer to	"Collecting Diagnostic Information" on page 27-11

DUMPCNT specifies the number of assertion failures for which one database server thread dumps shared memory or generates a core file by calling **gcore**. An assertion is a test of some condition or expression with the expectation that the outcome is true. For example, the following statement illustrates the concept of an assertion failure:

```
if (a != b)
    assert_fail("a != b");
```

UNIX

DUMPCORE

onconfig.std *value*	0
range of values	0 = Do not dump core. 1 = Dump core.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Dump Core
refer to	"Collecting Diagnostic Information" on page 27-11

DUMPCORE controls whether assertion failures cause a virtual processor to dump core. The core file is left in the directory from which the database server was last invoked. (The DUMPDIR parameter has no impact on the location of the core file.)

Warning: *When DUMPCORE is set to 1, an assertion failure causes a virtual processor to core dump, which in turn causes the database server to abort. DUMPCORE should be set only for debugging purposes in a controlled environment.*

DUMPDIR

onconfig.std *value*	**tmp**
if not present	**tmp**
range of values	Any directory to which user **informix** has write access
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Dump Directory
refer to	"Collecting Diagnostic Information" on page 27-11

DUMPDIR specifies a directory where the database server dumps shared memory, **gcore** files, or messages from a failed assertion. Because shared memory can be large, set DUMPDIR to a file system with a significant amount of space.

UNIX

DUMPGCORE

onconfig.std *value*	0
range of values	0 = Do not dump **gcore**. 1 = Dump **gcore**.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Dump Gcore
refer to	"Collecting Diagnostic Information" on page 27-11

DUMPGCORE is used with operating systems that support **gcore**. If you set DUMPGCORE, but your operating system does not support **gcore**, messages in the database server message log indicate that an attempt was made to dump core, but that the database server is unable to find the expected file. (If your operating system does not support **gcore**, set DUMPCORE instead.)

If DUMPGCORE is set, the database server calls **gcore** whenever a virtual processor encounters an assertion failure. The **gcore** utility directs the virtual processor to dump core to the directory specified by DUMPDIR and continue processing.

The core dump output generated by **gcore** is saved to the file **core.*pid.cnt***. The ***pid*** value is the virtual-processor process identification number. The ***cnt*** value is incremented each time this process encounters an assertion failure. The ***cnt*** value can range from 1 to the value of DUMPCNT. After that, no more core files are created. If the virtual processor continues to encounter assertion failures, errors are reported to the message log (and perhaps to the application), but no further diagnostic information is saved.

DUMPSHMEM

onconfig.std *value*	1
range of values	0 = Do not dump shared memory. 1 = Dump shared memory.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Dump Shared Memory
refer to	“Collecting Diagnostic Information” on page 27-11

DUMPSHMEM indicates that shared memory should be dumped on an assertion failure. All the shared memory used by the database server is dumped; it is probably quite large. The shared-memory dump is placed in a file in the directory specified by DUMPDIR.

The filename takes the format **shmem.*pid.cnt***. The ***pid*** value is the virtual-processor process identification number. The ***cnt*** value is incremented each time that this virtual processor encounters an assertion failure. The ***cnt*** value can range from 1 to the value of DUMPCNT. After the value of DUMPCNT is reached, no more files are created. If the database server continues to detect inconsistencies, errors are reported to the message log (and perhaps to the application), but no further diagnostic information is saved.

FILLFACTOR

onconfig.std *value*	90
units	Percent
range of values	1 through 100
takes effect	When the index is built. Existing indexes are not changed. To use the new value, the indexes must be rebuilt.
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Index Page Fill Factor
refer to	"Structure of B-Tree Index Pages" on page 38-48

FILLFACTOR specifies the degree of index-page fullness. A low value provides room for growth in the index. A high value compacts the index. If an index is full (100 percent), any new inserts result in splitting nodes. You can also set the FILLFACTOR as an option on the CREATE INDEX statement. The setting on the CREATE INDEX statement overrides the ONCONFIG file value.

HETERO_COMMIT

onconfig.std *value*	On Windows NT: 0
range of values	1 = Enable heterogeneous commit. 0 = Disable heterogeneous commit.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Heterogeneous Commit
refer to	"Heterogeneous Commit Protocol" on page 30-38 *INFORMIX-Enterprise Gateway Manager User Manual*

The HETERO_COMMIT configuration parameter specifies whether or not the database server is prepared to participate with Informix Gateway products in heterogeneous commit transactions. Setting HETERO_COMMIT to 1 allows a single transaction to update *one* non-Informix database (accessed using any of the Gateway products) and one or more Informix databases.

If HETERO_COMMIT is 0, a single transaction can update databases as follows:

- One or more Informix databases and no non-Informix database
- One non-Informix database and no Informix databases

You can read data from any number of Informix and non-Informix databases, regardless of the setting of HETERO_COMMIT.

LBU_PRESERVE

onconfig.std *value*	0
range of values	1 = Enable the logs-full high-water mark. 0 = Disable the logs-full high-water mark.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Preserve Log for Log Backup
refer to	"Enabling the Logs-Full High-Water Mark" on page 18-9

LBU_PRESERVE reserves the last logical log for administrative tasks by setting the logs-full high-water mark. When LBU_PRESERVE is enabled, the database server blocks further OLTP activity when the next-to-last log fills, rather than the last log. Setting LBU_PRESERVE prevents backups from failing due to lack of logical-log space.

LOCKS

onconfig.std *value*	2,000
range of values	2,000 through 8,000,000
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Max # of Locks
refer to	"Lock Table" on page 11-29; "How Database Server Threads Access Shared Buffers" on page 11-37

LOCKS specifies the maximum number of locks available to database server threads during processing. Although each additional lock takes up just 44 bytes of resident shared memory, locks can become a resource drain if you have a limited amount of shared memory. For example, if you set LOCKS to 1,000,000, the database server allocates 40 megabytes of resident shared memory for locks.

Tip: *When you drop a database, a lock is acquired and held on each table in the database until the database is dropped. For more information on the DROP DATABASE statement, see the "Informix Guide to SQL: Syntax."*

LOG_BACKUP_MODE

onconfig.std *value*	None
range of values	CONT = Continuous MANUAL = Manual
refer to	*INFORMIX-Enterprise Command Center User Guide* *Backup and Restore Guide*

The LOG_BACKUP_MODE parameter will be present only if you use INFORMIX-Enterprise Command Center (IECC) to turn on or turn off continuous backups of the logical-log files, and you use ON-Bar as your backup and restore tool. You must set the LOG_BACKUP_MODE parameter only through IECC.

IECC sets the LOG_BACKUP_MODE parameter to a value of `CONT` to turn on continuous logical-log backups. It sets LOG_BACKUP_MODE to `MANUAL` to turn off continuous logical-log backups.

The LOG_BACKUP_MODE parameter has the same effect as setting the ALARMPROGRAM parameter to the ALARMPROGRAM **onconfig.std** value when IECC is not installed.

LOGBUFF

onconfig.std *value*	32
units	Kilobytes
range of values	(2 * page size) through **LOGSIZE**
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Logical Log Buffer Size
refer to	"Logical-Log Buffer" on page 11-22

LOGBUFF specifies the size in kilobytes of each of the three logical-log buffers in shared memory. Triple buffering permits user threads to write to the active buffer while one of the other buffers is being flushed to disk. If flushing is not complete by the time the active buffer fills, the user thread begins writing to the third buffer.

Informix recommends that you set LOGBUFF to 16 or 32 kilobytes, or perhaps 64 kilobytes for heavy workloads.

System Page Size

The system page size is platform dependent. You can use the commands in the following table to display the system page size.

Command	Description
oncheck -pr	Checks the root-dbspace reserved pages and displays the system page size in the first section of its output
onstat -b	Displays the system page size, given as buffer size

UNIX

On UNIX, you can also use ON-Monitor to get the system page size under the Parameters:Shared-Memory option, which does not require the database server to be running, and under the Parameters:Initialize option. ♦

Important: *The database server uses the LOGBUFF parameter to set the size of internal buffers that are used during recovery. If you set LOGBUFF too high, the server can run out of memory and shut down during recovery.*

LOGFILES

onconfig.std value	On UNIX: 6 On Windows NT: 10
if not present	6
units	Number of logical-log files
range of values	3 through LOGSMAX (integers only)
takes effect	During disk initialization and when you use **onparams** to add a new log file. On UNIX, the value is also updated when you use ON-Monitor to add a new log file.
UNIX utilities	Parameters, Initialize, Number of Logical Logs **onparams**; see page 35-43.
Windows NT utilities	**onparams**; see page 35-43.
refer to	"What Should Be the Size and Number of Logical-Log Files?" on page 18-11 "Adding a Logical-Log File" on page 19-4 "Dropping a Logical-Log File" on page 19-6

LOGFILES specifies the number of logical-log files created by the database server during disk initialization. To change the number of logical-log files, add or drop a logical log files. If you use ON-Monitor (UNIX only) or **onparams** to add or drop log files, the database server automatically updates LOGFILES.

LOGSIZE

onconfig.std *value*	On UNIX: 1,500 On Windows NT: 500
if not present	On UNIX: 1,500 On Windows NT: 200
units	Kilobytes
range of values	200 through (ROOTSIZE -PHYSFILE - 512 - (63 * ((*pagesize*) / 1024))) / LOGFILES, where *pagesize* is the system page size
takes effect	When shared memory is initialized. The size of log files added after shared memory is initialized reflects the new value, but the size of existing log files does not change.
UNIX utilities	ON-Monitor: Parameters, Initialize, Log. Log Size
refer to	"How Big Should the Logical Log Be?" on page 18-5 "What Should Be the Size and Number of Logical-Log Files?" on page 18-11 "Changing LOGSIZE or LOGFILES" on page 19-10

LOGSIZE specifies the size that is used when logical-log files are created. It does not change the size of existing logical-log files. The total logical-log size is LOGSIZE * LOGFILES.

To verify the page size that the database server uses on your platform, see the last line of output from the **onstat -b** command.

LOGSMAX

onconfig.std *value*	On UNIX: 6 On Windows NT: 20
if not present	3
units	Number of logical-log files
range of values	LOGFILES through 32,767 (integers only)
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Max # of Logical Logs
refer to	"What Should Be the Size and Number of Logical-Log Files?" on page 18-11 "Moving a Logical-Log File to Another Dbspace" on page 19-8 "Changing Logical-Log Configuration Parameters" on page 19-10

LOGSMAX specifies the maximum number of logical-log files that an instance of the database server supports. The database server requires at least three logical-log files for operation.

In general, you can set the value of LOGSMAX equal to the value of LOGFILES. If you plan to relocate the logical-log files out of the root dbspace after you initialize the database server, assign LOGSMAX the value of LOGFILES plus 3.

LRUS

onconfig.std *value*	8
if not present	If MULTIPROCESSOR is set: MAX(4, NUMCPUVPS) If MULTIPROCESSOR is not set: 4
units	Number of LRU queues
range of values	For 64-bit platforms: 1 through 512 For all other platforms: 1 through 128
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Num of LRU queues
refer to	"LRU Queues" on page 11-37 *Performance Guide*

LRUS specifies the number of LRU (least-recently-used) queues in the shared-memory buffer pool. You can tune the value of LRUS, in combination with the LRU_MIN_DIRTY and LRU_MAX_DIRTY parameters, to control how frequently the shared-memory buffers are flushed to disk.

LRU_MAX_DIRTY

onconfig.std *value*	60
units	Percent
range of values	0 through 100
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, LRU Max Dirty
refer to	"LRU Queues" on page 11-37 "Limiting the Number of Pages Added to the MLRU Queues" on page 11-40

LRU_MAX_DIRTY specifies the percentage of modified pages in the LRU queues that, when reached, flags the queue to be cleaned.

LRU_MIN_DIRTY

onconfig.std *value*	50
units	Percent
range of values	0 through 100
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, LRU Min Dirty
refer to	"LRU Queues" on page 11-37 "When MLRU Cleaning Ends" on page 11-41

LRU_MIN_DIRTY specifies the percentage of modified pages in the LRU queues that, when reached, flags the page cleaners that cleaning is no longer mandatory. Page cleaners might continue cleaning beyond this point under some circumstances.

LTAPEBLK

onconfig.std *value*	16
units	Kilobytes
range of values	Values greater than (*page size* / 1024)
takes effect	For **ontape**: When you execute **ontape** For **onload** and **onunload**: When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Block Size (follows Log Tape Dev.)
refer to	*Archive and Backup Guide* for information on how this parameter affects **ontape**. For information on how this parameter affects **onload** and **onunload**, see the *Informix Migration Guide*.

LTAPEBLK specifies the block size of the device to which the logical logs are backed up when you use **ontape** for dbspace backups. LTAPEBLK also specifies the block size for the device to which data is loaded or unloaded when you use the **-l** option of **onload** or **onunload**. If you are using **onload** or **onunload**, you can specify a different block size at the command line.

Specify LTAPEBLK as the largest block size permitted by your tape device. The database server does not check the tape device when you specify the block size. Verify that the LTAPEDEV tape device can read the block size that you specify. If not, you might not be able to read from the tape.

UNIX

The UNIX **dd** utility can verify that the LTAPEDEV tape device can read the block size. It is available with most UNIX systems. ♦

System Page Size

The system page size is platform dependent. You can use the commands in the following table to display the system page size.

Command	Description
oncheck -pr	Checks the root-dbspace reserved pages and displays the system page size in the first section of its output
onstat -b	Displays the system page size, given as buffer size

UNIX

On UNIX, you can also use ON-Monitor to get the system page size under the Parameters:Shared-Memory option, which does not require the database server to be running, and under the Parameters:Initialize option. ♦

LTAPEDEV

onconfig.std *value*	On UNIX: **/dev/tapedev** On Windows NT: **\\.\TAPE1**
if not present	On UNIX: **/dev/null**
takes effect	For **ontape**: When you execute **ontape** For **onload** and **onunload**: When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Log Tape Dev
refer to	*Archive and Backup Guide* for information on how this parameter affects **ontape**. For information on how this parameter affects **onload** or **onunload**, see the *Informix Migration Guide*. "TAPEDEV" on page 33-88

LTAPEDEV specifies the device to which the logical logs are backed up when you use **ontape** for backups. LTAPEDEV also specifies to which device data is loaded or unloaded when you use the **-l** option of **onload** or **onunload**.

If you are using LTAPEDEV to specify a target device for **ontape**, read about setting and changing the value in the *Archive and Backup Guide*. If you are using LTAPEDEV to specify a device for **onunload** or **onload**, the same information for TAPEDEV is relevant for LTAPEDEV.

LTAPESIZE

onconfig.std *value*	10,240
units	Kilobytes
range of values	Positive integers
takes effect	For **ontape**: When you execute **ontape** For **onload** and **onunload**: When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Total Tape Size (follows Log Tape Dev.)
refer to	*Archive and Backup Guide* for information on how this parameter affects **ontape**. For information on how this parameter affects **onload** or **onunload**, see the *Informix Migration Guide*.

LTAPESIZE specifies the maximum tape size of the device to which the logical logs are backed up when you use **ontape** for backups. LTAPEDEV also specifies the maximum tape size of the device to which data is loaded or unloaded when you use the **-l** option of **onload** or **onunload**. If you are using **onload** or **onunload**, you can specify a different tape size on the command line.

LTXEHWM

onconfig.std *value*	60
units	Percent
range of values	LTXHWM through 100
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Long TX HWM Exclusive
refer to	"Avoiding Long Transactions" on page 18-18 "LTXHWM" on page 33-51

LTXEHWM specifies the *long-transaction, exclusive-access, high-water mark*. If the logical log fills to LTXEHWM, the long transaction currently being rolled back is given *exclusive* access to the logical log. For example, if you set LTXEHWM to 60 percent of the logical-log space, and the log fills to that percentage, at that point the thread that is rolling back the long transaction is given exclusive access to the logical log. The term *exclusive* is not entirely accurate. Most database server activity is suspended until the transaction has completed its rollback, but transactions that are in the process of rolling back or committing a transaction retain access to the logical log.

When the database server is initialized, if the value in the current configuration file is greater than 60, a warning is issued. Informix recommends that you change the value in your configuration file and increase the size of your logical-log files proportionately.

LTXHWM

onconfig.std *value*	50
units	Percent
range of values	1 through 100
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Long TX HWM
refer to	"Avoiding Long Transactions" on page 18-18

LTXHWM specifies the *long-transaction high-water mark*. The value of LTXHWM is the percentage of available logical-log space that, when filled, triggers the database server to check for a long transaction. If a long transaction is found, the transaction is aborted, and the database server rolls back all modifications associated with it. Other transactions continue to execute, and the rollback procedure itself generates logical-log records, so the logical log continues to fill. (The LTXEHWM parameter exists for this reason.)

When the database server is initialized, if the value in the current configuration file is greater than 50, a warning is issued. Informix recommends that you change the value in your configuration file and increase the size of your logical-log files proportionately.

MAX_PDQPRIORITY

onconfig.std *value*	100
if not present	100
range of values	0 through 100
takes effect	On all user sessions
UNIX utilities	ON-Monitor: Parameters menu, pdQ screen **onmode -D**; see page 35-39.
Windows NT utilities	**onmode -D**; see page 35-39.
refer to	*Performance Guide*

MAX_PDQPRIORITY limits the PDQ resources that the database server can allocate to any one DSS query. MAX_PDQPRIORITY is a factor that is used to scale the value of PDQ priority set by users. For example, suppose that the database administrator sets MAX_PDQPRIORITY to 80. If a user sets the **PDQPRIORITY** environment variable to 50 and then issues a query, the database server silently processes the query with a PDQ priority of 40.

MAX_PDQPRIORITY lets the database administrator run decision support concurrently with OLTP without a deterioration of OLTP performance. However, if MAX_PDQPRIORITY is too low, the performance of decision support queries can degrade.

MAX_PDQPRIORITY takes one of the following values and affects all user sessions.

Value	Description
0	PDQ is turned off; DSS queries use no parallelism.
1	The database server fetches data from fragmented tables in parallel (parallel scans) but uses no other form of parallelism.
100	Use all available resources for processing queries in parallel.
number	The *number* is an integer number between 0 and 100 that indicates the percentage of the user requested PDQ resources that the database server actually allocates to the query. Resources include the memory, CPU, disk I/O, and scan threads.

MIRROR

onconfig.std *value*	0
range of values	0 = Disable mirroring. 1 = Enable mirroring.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Mirror
refer to	“Enabling Mirroring” on page 24-4

MIRROR is a flag that indicates whether mirroring is enabled for the database server. Enable mirroring if you plan to create a mirror for the root dbspace as part of initialization. Otherwise, leave mirroring disabled.

MIRROROFFSET

onconfig.std *value*	0
units	Kilobytes
range of values	Any value greater than or equal to 0
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Offset (below Mirror Path)
refer to	"Mirroring the Root Dbspace During Initialization" on page 24-7

MIRROROFFSET specifies the offset into the disk partition or into the device to reach the chunk that serves as the mirror for the initial chunk of the root dbspace.

MIRRORPATH

onconfig.std *value*	None
range of values	Pathname
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Mirror Path
refer to	"Mirroring the Root Dbspace During Initialization" on page 24-7

MIRRORPATH specifies the full pathname of the chunk that serves as the mirror for the initial chunk of the root dbspace. MIRRORPATH should be a link to the chunk pathname of the actual mirrored chunk for the same reasons that ROOTPATH is specified as a link. Similarly, select a short pathname for the mirrored chunk.

MSGPATH

onconfig.std *value*	On UNIX: **/usr/informix/online.log** On Windows NT: **online.log**
if not present	On UNIX: **/dev/tty**
range of values	Pathname
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, diaGnostics, Message Log
refer to	"What Is the Message Log?" on page 29-7

MSGPATH specifies the full pathname of the message-log file. The database server writes status messages and diagnostic messages to this file during operation.

If the file specified by MSGPATH does not exist, the database server creates the file in the specified directory. If the directory specified by MSGPATH does not exist, the database server sends the messages to the database administrator's terminal.

If the file specified by MSGPATH does exist, the database server opens it and appends messages to it as they occur.

MULTIPROCESSOR

onconfig.std *value*	0
if not present	Platform dependent.
range of values	0 = No multiprocessor 1 = Multiprocessor available
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Multiprocessor Machine
refer to	"CPU Virtual Processors" on page 9-20

MULTIPROCESSOR specifies whether the database server performs locking in a manner that is suitable for a single-processor computer or a multiprocessor computer.

NETTYPE

syntax

On UNIX or Windows NT:

```
NETTYPE protocol,poll_threads,connections,VP_class
```

On UNIX:

```
NETTYPE SQLMUX
```

The *protocol* value is required. You cannot use any white space in the fields, but you can omit trailing commas.

***onconfig.std* values**

On UNIX: None
On Windows NT: onsoctcp,1,,NET

if not present

protocol:
On UNIX: **nettype** field from the **sqlhosts** file or registry (optionally minus the database server prefix of **on** or **ol**)
On Windows NT: onsoctcp

poll_threads: 1

connections: 50

VP_class: NET if it is for DBSERVERALIASES; CPU if it is for DBSERVERNAME

units

poll_threads: NUMCPUVPS

connections: Number of connections

separators

Commas

range of values	*protocol*: On UNIX: same as the **nettype** values (with or without the database server prefix of **on** or **ol**) that are accepted in the **sqlhosts** file or registry On Windows NT: onsoctcp *poll_threads*: On UNIX if *VP_class* is NET: A value greater than or equal to 1 On UNIX if *VP_class* is CPU: 1 through NUMCPUVPS On Windows NT: Any value greater than or equal to 1 *connections*: 1 through 32767 *VP_class*: CPU = CPU VPs NET = Network VPs To enable multiplexed connections on UNIX, use **SQLMUX** as the only parameter value.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters: perFormance, Protocol, Threads, Users (*connections*), VP-class entries for the protocols supported on the computer
refer to	"What Is a Multiplexed Connection?" on page 4-7 "The nettype and Protocol Field" on page 4-31 "Network Virtual Processors" on page 9-28 "Should Poll Threads Run on CPU or Network Virtual Processors?" on page 9-29 "Multiplexed Connections" on page 33-60

Except for the NETTYPE parameter that lets the database server implement multiplexed connections on UNIX, the NETTYPE parameter provides tuning options for the protocols defined by **dbservername** entries in the **sqlhosts** file or registry.

Each **dbservername** entry in the **sqlhosts** file or registry is defined on either the DBSERVERNAME parameter or the DBSERVERALIASES parameter in the ONCONFIG file.

The NETTYPE configuration parameter describes a network connection as follows:

- The protocol (or type of connection)
- The number of poll threads assigned to manage the connection
- The expected number of concurrent connections
- The class of virtual processor that will run the poll threads

You can specify a NETTYPE parameter for each protocol that you want the database server to use. The following example illustrates NETTYPE parameters for two types of connections to the database server, a shared memory connection for local clients and a network connection that uses sockets:

```
NETTYPE ipcshm,3,,CPU
NETTYPE soctcp,,20,NET
```

The NETTYPE parameter for the shared-memory connection (ipcshm) specifies three poll threads to run in CPU virtual processors. The number of connections is not specified, so it is set to 50. The NETTYPE parameter for the sockets connection (soctcp) specifies that only 20 simultaneous connections are expected for this protocol and that one (because the number of poll threads is not specified) poll thread will run in a network virtual processor (in this case, SOC).

Protocol

The protocol entry is the same as the **nettype** field in the **sqlhosts** file or registry, except that the database server prefix of **on** or **ol** is optional. The first three characters of the protocol entry specify the interface type, and the last three characters specify the IPC mechanism or the network protocol.

Number of Poll Threads

If your database server has a large number of connections, you might be able to improve performance by increasing the number of poll threads. In general, each poll thread can handle approximately 200 to 250 connections.

The default value of *connections* is 50. This field specifies the maximum number of connections that can use this protocol at the same time. If you know that only a few connections will be using a protocol concurrently, you might save memory by explicitly setting the estimated number of connections.

For all net types other than ipcshm, the poll threads dynamically reallocate resources to support more connections as needed. Avoid setting the value for the number of concurrent connections to much higher than you expect. Otherwise, you might waste system resources.

Class of Virtual Processor

You can set the *VP_class* entry to specify either CPU or NET. However, the combined number of poll threads defined with the CPU VP class for all net types cannot exceed the maximum number of CPU VPS.

Default Values

Informix recommends that you use NETTYPE to configure each of your connections. However, if you do not use NETTYPE, the database server uses the default values to create a single poll thread for the protocol. If the dbservername is defined by DBSERVERNAME, by default the poll thread is run by the CPU class. If the dbservername is defined by DBSERVERALIASES, the default VP class is NET.

Multiplexed Connections

To enable the database server to use multiplexed connections, you must include a special NETTYPE parameter with the value SQLMUX, as shown in the following example:

```
NETTYPE SQLMUX
```

NOAGE

onconfig.std *value*	0
range of values	0 = Use priority aging. 1 = Disable priority aging.
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Disable Priority Aging
refer to	"Preventing Priority Aging" on page 9-22

Some operating systems lower the priority of processes as the processes run over a long period of time. NOAGE, when set to 1, disables *priority aging* of CPU virtual processors by the operating system. When NOAGE is set to the default, 0, the operating system might lower the priority of CPU virtual processors, as well as other processes, as they accumulate processing time. If your operating system supports priority aging, Informix recommends that you set NOAGE to 1.

NUMAIOVPS

onconfig.std *value*	On UNIX: None On Windows NT: 1
if not present	max((2 * *number_of_chunks*), 6), where *number_of_chunks* is the number of chunks that you have allocated
units	Number of AIO VPs
range of values	Integer greater than or equal to 1
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, AIO VPs **onmode -p**; see page 35-37.
Windows NT utilities	**onmode -p**; see page 35-37.
refer to	"Asynchronous I/O" on page 9-26

NUMAIOVPS specifies the number of virtual processors of the AIO class to run. Unless kernel asynchronous I/O is implemented, the AIO virtual processors perform all the database server disk I/O, other than I/O to the log files.

NUMCPUVPS

onconfig.std *value*	1
units	Number of CPU VPs
range of values	1 through the number of CPUs
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, CPU VPs **onmode -p**; see page 35-37.
Windows NT utilities	**onmode -p**; see page 35-37.
refer to	"CPU Virtual Processors" on page 9-20

NUMCPUVPS specifies the number of virtual processors of the CPU class to run. CPU virtual processors run all threads that start as the result of a connection by a client application, as well as internal threads. On a single-processor computer, allocate only one CPU virtual processor. On a multiprocessor computer, do not allocate more CPU virtual processors than there are CPUs on the computer.

OFF_RECVRY_THREADS

onconfig.std *value*	10
units	Number of recovery threads
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Off-Line Recovery Threads
refer to	*Backup and Restore Guide* for ON-Bar and *Archive and Backup Guide* for ON-Archive or **ontape**

OFF_RECVRY_THREADS is the number of recovery threads used in logical recovery when the database server is off-line (during a cold restore). This number of threads is also used to roll forward logical-log records in fast recovery.

Before you perform a cold restore, you can set the value of this parameter to approximately the number of tables that have a large number of transactions against them in the logical log. For single-processor computers, more than 30 to 40 threads is probably too many because the increase in parallel processing is probably offset by the overhead of thread management.

ON_RECVRY_THREADS

onconfig.std *value*	1
units	Number of recovery threads
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, On-Line Recovery Threads
refer to	*Backup and Restore Guide* for ON-Bar. See your *Archive and Backup Guide* for ON-Archive or **ontape**.

ON_RECVRY_THREADS is the maximum number of recovery threads that the database server uses for logical recovery when the database server is on-line (that is, during a warm restore).

You can tune ON_RECVRY_THREADS to the number of tables that are likely to be recovered because the logical-log records that are processed during recovery are assigned threads by table number. The maximum degree of parallel processing occurs when the number of recovery threads matches the number of tables being recovered.

ONDBSPACEDOWN

onconfig.std *value*	On UNIX: 0 On Windows NT: 2
range of values	0 = Continue. Causes the database server to mark a noncritical dbspace down and continue whenever a disabling I/O error occurs on it. 1 = Abort. Causes the database server to crash without allowing a checkpoint to occur whenever a disabling I/O error occurs on any dbspace. Critical dbspaces run only in this mode. 2 = Wait. Causes the database server to hang all updating threads as soon as the next checkpoint request occurs after a disabling I/O occurs on a noncritical dbspace.
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Dbspace Down Option
refer to	"Monitoring the Database Server for Disabling I/O Errors" on page 29-52

ONDBSPACEDOWN defines the action that the database server will take when any disabling event occurs on a noncritical dbspace.

UNIX

OPCACHEMAX

onconfig.std *value*	0
if not present	128
units	Kilobytes
range of values	0 through (4 * 1024 * 1024)
takes effect	When the Optical Subsystem needs more memory
UNIX utilities	ON-Monitor: Parameters, Initialize, StageBlob
refer to	*Guide to the Optical Subsystem*

OPCACHEMAX specifies the size of the memory cache for the Optical Subsystem. The database server stores pieces of TEXT or BYTE data types in the memory cache before it delivers them to the subsystem. Use this parameter only if you use the Optical Subsystem.

OPTCOMPIND

onconfig.std *value*	2
range of values	0 = When appropriate indexes exist for each ordered pair of tables, the optimizer chooses index scans (nested-loop joins), without consideration of the cost, over table scans (sort-merge joins or hash joins). This value ensures compatibility with previous versions of the database server. 1 = If the transaction isolation mode is not Repeatable Read, the optimizer behaves as it does for the value 2. Otherwise, the optimizer behaves as it does for the value 0. Informix recommends this setting for optimal performance. 2 = The optimizer uses costs to determine an execution path regardless of the transaction-isolation mode. Index scans (nested-loop joins) are not given preference over table scans (other join methods); the optimizer bases its decision purely on costs. This value is the default if the variable is not set.
UNIX utilities	ON-Monitor: Parameters menu, pdQ screen
refer to	*Performance Guide* The **OPTCOMPIND** environment variable in the *Informix Guide to SQL: Reference*

OPTCOMPIND helps the optimizer choose an appropriate access method for your application.

Tip: *You can think of the name of the variable as arising from "OPTimizer COMPare (the cost of using) INDexes (with other methods)."*

Because of the nature of *hash joins* and *sort-merge*, an application with isolation mode set to Repeatable Read might *temporarily* lock all records in tables that are involved in the join (even those records that fail to qualify the join) for each ordered set of tables. This situation leads to higher contention among connections. Conversely, nested-loop joins lock fewer records but provide inferior performance when the database server retrieves a large number of rows. Thus, both join methods offer advantages and disadvantages. A client application can also influence the optimizer in its choice of a join method.

PHYSBUFF

onconfig.std *value*	32
units	Kilobytes
range of values	(Page size) through PHYSFILE
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Physical-log Buffer Size
refer to	"Physical-Log Buffer" on page 11-24

PHYSBUFF specifies the size in kilobytes of each of the two physical-log buffers in shared memory. Double buffering permits user threads to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. The value of the PHYSBUFF parameter determines how frequently the database server needs to flush the physical-log buffer to the physical-log file.

A write to the physical-log buffer is exactly one page in length. Choose a value for PHYSBUFF that is evenly divisible by the page size. If the value of PHYSBUFF is not evenly divisible by the page size, the database server rounds down the size to the nearest value that is evenly divisible by the page size.

The minimum value for PHYSBUFF is the size of one page. The recommended value for PHYSBUFF is 16 pages.

System Page Size

The system page size is platform dependent. You can use the commands in the following table to display the system page size.

Command	Description
oncheck -pr	Checks the root-dbspace reserved pages and displays the system page size in the first section of its output
onstat -b	Displays the system page size, given as buffer size

UNIX

On UNIX, you can also use ON-Monitor to get the system page size under the Parameters:Shared-Memory option, which does not require the database server to be running, and under the Parameters:Initialize option. ♦

PHYSDBS

***onconfig.std* value**	**rootdbs**
if not present	ROOTNAME
units	A dbspace
range of values	Up to 18 characters
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Physical-Log, Dbspace Name
refer to	"Where Is the Physical Log Located?" on page 20-8 and "Changing the Physical-Log Location and Size" on page 21-3

PHYSDBS specifies the name of the dbspace that contains the physical log. You can move the physical log to a dbspace other than the root dbspace to reduce disk contention.

PHYSFILE

onconfig.std *value*	On UNIX: 1000 On Windows NT: 2000
if not present	200
units	Kilobytes
range of values	200 or more
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Init, Phys. Log Size
refer to	"How Big Should the Physical Log Be?" on page 20-5 and "Changing the Physical-Log Location and Size" on page 21-3

PHYSFILE specifies the size of the physical log.

RA_PAGES

onconfig.std *value*	None
if not present	If MULTIPROCESSOR is 0: 4 If MULTIPROCESSOR is 1: 8
units	Number of pages
range of values	RA_THRESHOLD through BUFFERS
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Num of Read Ahead Pages
refer to	"Configuring the Database Server to Read Ahead" on page 11-42

RA_PAGES specifies the number of disk pages to attempt to read ahead during sequential scans of data or index records. Read-ahead can greatly speed up database processing by compensating for the slowness of I/O processing relative to the speed of CPU processing.

This parameter works with the RA_THRESHOLD parameter. Specifying values that are too large can result in excessive buffer-caching activity. For more information on calculating RA_PAGES and RA_THRESHOLD, see your *Performance Guide*.

RA_THRESHOLD

onconfig.std *value*	None
if not present	RA_PAGES / 2
units	Number of pages
range of values	0 through (RA_PAGES-1)
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Read Ahead Threshold
refer to	"Configuring the Database Server to Read Ahead" on page 11-42

RA_THRESHOLD is used with RA_PAGES when the database server reads during sequential scans of data and index records. RA_THRESHOLD specifies the read-ahead threshold; that is, the number of unprocessed pages in memory that signals the database server to perform the next read-ahead.

Specifying values that are too large for RA_PAGES and RA_THRESHOLD can result in excessive buffer-caching activity. For more information on calculating RA_PAGES and RA_THRESHOLD, see your *Performance Guide*.

RESIDENT

onconfig.std *value*	0
range of values	0 = Off -1 = Make all resident and virtual segments resident *number*, where *number* > 0 = Make *number* segments resident
if not present	0
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Forced Residency **onmode**; see page 35-28.
Windows NT utilities	**onmode**; see page 35-28.
refer to	"The Resident Portion of Shared Memory" on page 11-18

The RESIDENT parameter specifies whether resident and virtual segments of shared memory remain resident in operating-system physical memory. If your operating system supports forced residency, you can specify that resident and virtual segments of shared memory not be swapped to disk.

Warning: *The size of shared memory is a factor in your decision. Before you decide on residency, verify that, after satisfying the database-server requirements, the amount of physical memory available is sufficient to execute all required operating-system and application processes.*

RESTARTABLE_RESTORE

onconfig.std *value*	0
range of values	0 = No 1 = Yes
takes effect	When shared memory is initialized
refer to	*Backup and Restore Guide*

If you set RESTARTABLE_RESTORE to 1, you simply enable the database server to restart a failed physical or logical restore close to the point at which the failure occurred. You can only perform a restartable if you use the ON-Bar backup and restore tool.

The database server uses physical recovery and logical recovery to restore data from tape:

- Physical recovery

  The database server writes dbspace pages from the backup tape to disk. This action leaves the dbspace consistent to the point in time at which it was originally backed up. However, the point in time is different for each dbspace managed by the database server, because the backup times are usually different. A restartable restore is restartable to the level of a dbspace. If only some chunks of a dbspace are restored when the restore fails, the entire dbspace needs to be recovered again when you restart the restore.

- Logical recovery

  The database server replays records from logical log files on tape to bring data in all of the dbspaces up to date. At the end of logical recovery, all dbspaces are consistent to the same point in time.

Important: *If the database server crashes in the middle of a warm logical restore, you must restart the physical restore, and then restart the logical restore.*

For more information on the requirements and procedures to restart a failed restore, refer to your *Backup and Restore Guide*.

ROOTNAME

onconfig.std *value*	**rootdbs**
units	A dbspace
range of values	Up to 18 characters
takes effect	When disk is initialized (destroys all data)
UNIX utilities	ON-Monitor: Parameters, Initialize, Root Name
refer to	"Allocating Disk Space" on page 14-4

ROOTNAME specifies a name for the root dbspace for this database server configuration. The name must be unique among all dbspaces and blobspaces managed by this instance of the database server. The name cannot exceed 18 characters. Valid characters are restricted to digits, letters, and the underscore. Informix recommends that you select a name that is easily recognizable as the root dbspace.

ROOTOFFSET

onconfig.std *value*	0
units	Kilobytes
range of values	Any value greater than or equal to 0
takes effect	When disk is initialized (destroys all data)
UNIX utilities	ON-Monitor: Parameters, Initialize, Offset (follows Root Size)
refer to	"Allocating Raw Disk Space on UNIX" on page 14-8

ROOTOFFSET specifies the offset into an allocation of disk space (file, disk partition, or device) at which the initial chunk of the root dbspace begins.

ROOTPATH

onconfig.std *value*	On UNIX: **/dev/dynsrv_root** On Windows NT: None
range of values	Pathname
takes effect	When disk is initialized (destroys all data)
UNIX utilities	ON-Monitor: Parameters, Initialize, Primary Path
refer to	"Allocating Disk Space" on page 14-4

ROOTPATH specifies the full pathname, including the device or filename, of the initial chunk of the root dbspace. ROOTPATH is stored in the reserved pages as a chunk name. You must set the permissions of the file specified in ROOTPATH to 660. The owner and group must both be **informix**.

UNIX

If you use unbuffered disk space for your initial chunk on UNIX, Informix recommends that, instead of entering the actual device name for the initial chunk, you define ROOTPATH as a pathname that is a link to the root dbspace initial chunk. "Creating a Link to Each Raw Device on UNIX" on page 14-9 presents a rationale for using links rather than device names. ♦

ROOTSIZE

onconfig.std *value*	20,000
if not present	0
units	Kilobytes
range of values	0 through (maximum capacity of the storage device)
takes effect	When disk is initialized (destroys all data)
UNIX utilities	ON-Monitor: Parameters, Initialize, Root Size
refer to	"Calculate the Size of the Root Dbspace" on page 13-30

ROOTSIZE specifies the size of the initial chunk of the root dbspace, expressed in kilobytes. To change ROOTSIZE after you initialize the database server, completely unload and reload your data.

SERVERNUM

onconfig.std *value*	0
range of values	0 through 255
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Server Number
refer to	"The Role of the SERVERNUM Configuration Parameter" on page 5-5

SERVERNUM specifies a relative location in shared memory. The value that you choose is not important, but it must be unique for each database server on your local host computer. The value does not need to be unique on your network. Informix recommends that you choose a value other than 0 to avoid inadvertent duplication of SERVERNUM.

SHMADD

onconfig.std *value*	8192
units	Segment size
range of values	1024 through 524,288
units	Bytes
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Add Seg Size
refer to	"The Virtual Portion of Shared Memory" on page 11-25

SHMADD specifies the size of a segment that is dynamically added to the virtual portion of shared memory. It is more efficient to add memory in large segments, but it is wasteful if the added memory is not used. Also, the operating system might require you to add memory in a few large segments rather than many small segments.

Informix recommends the guidelines in the following table for setting the initial value of SHMADD.

Amount of Physical Memory	Recommended SHMADD Value
Less than 256 megabytes	8192
Greater than 256 megabytes and less than 512 megabytes	16,384
Greater than 512 megabytes	32,768

The following command displays the size of memory segments and the amount of memory that is used or free in each:

```
onstat -g seg
```

SHMBASE

onconfig.std *value*	On UNIX: Platform dependent On Windows NT: 0xC000000L
units	Address
range of values	Positive integers
takes effect	When shared memory is initialized
refer to	"Setting Operating-System Shared-Memory Configuration Parameters" on page 12-3

SHMBASE specifies the base address where shared memory is attached to the memory space of a virtual processor. Do not change the value of SHMBASE. The **onconfig.std** value for SHMBASE is platform dependent.

UNIX

ON-Monitor, which is available only on UNIX, does not prompt for this value during initialization. ♦

SHMTOTAL

onconfig.std *value*	0
units	Kilobytes
range of values	Integer greater than or equal to 1
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Total Memory
refer to	"How Much Shared Memory Does the Database Server Need?" on page 11-9

SHMTOTAL specifies the total amount of shared memory (resident, virtual, communications, and virtual extension portions of shared memory) to be used by the database server for all memory allocations. The **onconfig.std** value of 0 implies that no limit on memory allocation is stipulated.

SHMTOTAL enables you to limit the demand for memory that the database server can place on your system. However, applications might fail if the database server requires more memory than the limit imposed by SHMTOTAL. When this situation occurs, the database server writes the following message in the message log:

```
size of resident + virtual segments xx + yy > zz total allowed by configuration
parameter SHMTOTAL
```

This message includes the following values.

Value	Description
xx	Current size of resident segments
yy	Current size of virtual segments
zz	Total shared-memory required

SHMVIRTSIZE

onconfig.std *value*	On UNIX: 8000 On Windows NT: 8192
if not present	If SHMADD is present: SHMADD If SHMADD is not present: 8
units	Kilobytes
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Non Res. Seg Size
refer to	"The Virtual Portion of Shared Memory" on page 11-25

SHMVIRTSIZE specifies the initial size of a virtual shared-memory segment. Use the following algorithm to determine the size of the virtual portion of shared memory:

```
fixed_overhead + shared_structures + (mncs * private_structures)
```

This algorithm includes the following values.

Value	Description
fixed_overhead	global pool + thread pool after booting (partially dependent on the number of virtual processors
shared_structures	AIO vectors + sort memory + dbspace backup buffers + dictionary size + size of stored procedure cache + histogram pool + other pools (see the **onstat** display)
mncs	maximum number of concurrent sessions
private_structures	stack (generally 32 kilobytes but dependent on recursion in stored procedures and triggers) + heap (about 30 kilobytes) + session control-block structures

If messages in the message file indicate that the database server is adding segments to the virtual portion of shared memory for you, add the amount indicated by these messages to the value of SHMVIRTSIZE. Informix recommends that you initially create a virtual portion of shared memory of a size that is more than sufficient for your daily processing, if possible.

The following command determines peak usage and lowers the value of SHMVIRTSIZE accordingly:

```
% onstat -g seg
```

SINGLE_CPU_VP

onconfig.std *value*	0
range of values	0 = Running with multiple CPU VPs Any nonzero value = Running with one CPU VP
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, perFormance, Single CPU VP
refer to	"Running on a Single-Processor Computer" on page 9-21

SINGLE_CPU_VP specifies whether or not the database server is running with only one CPU virtual processor.

When SINGLE_CPU_VP is nonzero, the database server can use code that is optimized for a single CPU virtual processor. This arrangement lets it bypass many of the mutex calls that it must use when it runs multiple CPU virtual processors.

If you set this parameter to nonzero and then attempt to bring up the database server with NUMCPUVPS set to a value greater than 1, you receive the following error message, and the database-server initialization fails:

```
Cannot have SINGLE_CPU_VP non-zero and CPU VPs greater than 1.
```

If you set SINGLE_CPU_VP to nonzero and try to add a CPU virtual processor, you receive one of the following messages:

```
onmode: failed when trying to change the number of classname VPs by n.
onmode: failed when trying to change the number of cpu virtual processors by n.
```

Informix strongly recommends that you set this parameter when the database server will run only one CPU virtual processor. Depending on the application and work load, setting this parameter can improve performance by up to 10 percent.

STACKSIZE

***onconfig.std* value**	32
units	Kilobytes
range of values	32 through (limit determined by the database server configuration and the amount of memory available)
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Stack Size (kilobytes)
refer to	"Stacks" on page 9-15 The CREATE FUNCTION statement in the *Informix Guide to SQL: Syntax*

The STACKSIZE parameter specifies the stack size for the database server user threads. The value of STACKSIZE does not have an upper limit, but setting a value that is too large wastes virtual memory space and can cause swap-space problems.

Informix has determined that the default size of 32 kilobytes is sufficient for nonrecursive database activity. When the database server performs recursive database tasks, as in some stored procedures, for example, it explicitly checks for the possibility of stack-size overflow and automatically expands the stack.

Warning: *Setting the value of STACKSIZE too low can cause stack overflow, the result of which is undefined but usually undesirable.*

STAGEBLOB

onconfig.std *value*	None
range of values	Up to 18 characters
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Stage Blob
refer to	*Guide to the Optical Subsystem*

Use this parameter only if you are storing TEXT or BYTE data types on optical storage with the Optical Subsystem. This parameter has no affect on ordinary blobspaces.

STAGEBLOB is the blobspace name for the area where the Optical Subsystem stages TEXT and BYTE data types that are destined for storage on optical disk.

TAPEBLK

onconfig.std *value*	16
units	Kilobytes
range of values	Values greater than (page size) / 1024
takes effect	For **ontape**: When you execute **ontape** For **onload** and **onunload**: When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Block Size (follows Tape Dev)
refer to	*Informix Migration Guide*

TAPEBLK specifies the block size of the device to which **ontape** writes during a dbspace backup. TAPEBLK also specifies the default block size of the device to which data is loaded or unloaded when you use **onload** or **onunload**. If you are using **onload** or **onunload**, you can specify a different block size on the command line.

The database server does not check the tape device when you specify the block size. Verify that the TAPEBLK tape device can read the block size that you specify. If not, you might not able to read from the tape.

System Page Size

The system page size is platform dependent. You can use the commands in the following table to display the system page size.

Command	Description
oncheck -pr	Checks the root-dbspace reserved pages and displays the system page size in the first section of its output
onstat -b	Displays the system page size, given as buffer size

UNIX

On UNIX, you can also use ON-Monitor to get the system page size under the Parameters:Shared-Memory option, which does not require the database server to be running, and under the Parameters:Initialize option. ♦

TAPEDEV

onconfig.std *value*	On UNIX: **/dev/tapedev** On Windows NT: **\\.\TAPE0**
if not present	On UNIX: **/dev/null**
units	Pathname
takes effect	For **ontape**: When you execute **ontape** For **onload** and **onunload**: When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Tape Dev
refer to	*Archive and Backup Guide* for information on how this parameter affects **ontape**. For information on how this parameter affects **onload** and **onunload**, see the *Informix Migration Guide*.

TAPEDEV specifies the device to which **ontape** back ups dbspace data. TAPEDEV also specifies the default device to which data is loaded or unloaded when you use **onload** or **onunload**.

UNIX

Using Symbolic Links

TAPEDEV can be a symbolic link, enabling you to switch between tape devices without changing the pathname specified in TAPEDEV.

UNIX

Using a Remote Device

You can use the following syntax to specify a tape device attached to another host computer:

```
host_machine_name:tape_device_pathname
```

The following example specifies a tape device on the host computer **kyoto**:

```
kyoto:/dev/rmt01
```

Verify the Tape Block Size and Tape Size

If you change the tape device, verify that TAPEBLK and TAPESIZE are correct for the new device.

Tape Devices Must Rewind Before Opening and on Closing

The tape device specified by TAPEDEV must perform a rewind before it opens and when it closes. The database server requires this action because of a series of checks that it performs before it writes to a tape.

When the database server attempts to *write* to any tape other than the first tape in a multivolume dbspace or logical-log backup, the database server first reads the tape header to make sure that the tape is available for use. Then the device is closed and reopened. The database server assumes the tape was rewound when it closed, and the database server begins to write.

Whenever the database server attempts to *read* a tape, it first reads the header and looks for the correct information. The database server does not find the correct header information at the start of the tape if the tape device did not rewind when it closed during the write process.

TAPESIZE

onconfig.std *value*	10,240
units	Kilobytes
range of values	Positive integers
takes effect	For **ontape**: When you execute **ontape** For **onload** and **onunload**: When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Initialize, Total Tape Size (follows Tape Dev)
refer to	*Informix Migration Guide*

The TAPESIZE parameter specifies the size of the device to which **ontape** backs up dbspace data. TAPESIZE also specifies the size of the default device to which data is loaded or unloaded when you use **onload** or **onunload**. If you are using **onload** or **onunload**, you can specify a different tape size on the command line.

TXTIMEOUT

onconfig.std *value*	300
units	Seconds
range of values	Positive integers
takes effect	When shared memory is initialized
UNIX utilities	ON-Monitor: Parameters, Shared-Memory, Transaction Timeout
refer to	"How the Two-Phase Commit Protocol Handles Failures" on page 30-10

TXTIMEOUT specifies the amount of time that a participant in a two-phase commit waits before it initiates participant recovery.

This parameter is used only for distributed queries that involve a remote database server. Nondistributed queries do not use this parameter.

USEOSTIME

onconfig.std *value*	0
range of values	0 = Off 1 = On
takes effect	During initialization
UNIX utilities	ON-Monitor: Parameters, perFormance, Use OS Time
refer to	*Performance Guide*

Setting USEOSTIME to 1 specifies that the database server is to use subsecond granularity when it obtains the current time from the operating system for SQL statements. If subsecond granularity is not needed, the database server retrieves the current time from the operating system once per second, making the granularity of time for client applications one second. When the host computer for the database server has a clock with subsecond granularity, applications that depend on subsecond accuracy for their SQL statements should set USEOSTIME to 1.

Systems that run with USEOSTIME set to nonzero notice a performance degradation of up to 4 to 5 percent versus running with USEOSTIME turned off.

This setting does not affect any calls regarding the time from application programs to Informix embedded-language library functions.

Chapter

34

The sysmaster Database

What Is the sysmaster Database? 34-3

Using the System-Monitoring Interface 34-5
- What Are the SMI Tables? 34-5
- Accessing SMI Tables. 34-6
  - SELECT Statements 34-6
  - Triggers and Event Alarms 34-7
  - SPL and SMI Tables 34-7
  - Locking and SMI Tables 34-7

The System-Monitoring Interface Tables 34-8
- sysadtinfo 34-9
- sysaudit 34-10
- syschkio 34-11
- syschunks 34-11
- sysconfig 34-13
- sysdatabases. 34-14
- sysdbslocale 34-15
- sysdbspaces 34-15
- sysdri 34-17
- sysextents. 34-18
- syslocks 34-18
- syslogs. 34-19
- sysprofile 34-20
- sysptprof 34-23
- syssesprof 34-24
- syssessions 34-26
- sysseswts 34-28
- systabnames. 34-29
- sysvpprof. 34-30

The SMI Tables Map 34-30
Using SMI Tables to Obtain onstat Information 34-33

This chapter describes the **sysmaster** database and contains reference information for the *system-monitoring interface* (SMI). It provides information on the following topics:

- How to use SMI tables
- Descriptions of the SMI tables
- A map of the documented SMI tables

The **sysmaster** database also contains information about ON-Archive catalog tables. For information about ON-Archive, see your *Archive and Backup Guide*.

What Is the sysmaster Database?

The database server creates and maintains the **sysmaster** database. It is analogous to the system catalog for databases, which is described in the *Informix Guide to SQL: Reference*. Just as a system catalog for every database managed by the database server keeps track of objects and privileges in the database, a **sysmaster** database for every database server keeps track of information about the database server.

The **sysmaster** database contains the following tables:

- SMI tables

  The *system-monitoring interface* tables in the **sysmaster** database provide information about the state of the database server. You can query these tables to identify processing bottlenecks, determine resource usage, track session or database server activity, and so on. This chapter describes these tables, which are slightly different than ordinary tables.

- ON-Archive catalog tables

  The ON-Archive catalog tables store information about ON-Archive requests, volume sets, and save sets.

Warning: *The database server relies on information in the **sysmaster** database. Do not change any of the tables in **sysmaster** or any of the data within the tables. Such changes could cause unpredictable and debilitating results.*

The database server creates the **sysmaster** database when it initializes disk space. The database server creates the database with unbuffered logging. You cannot drop the database or any of the tables in it, and you cannot turn logging off.

As user **dddd**, you can create stored procedures in the **sysmaster** database. (You can also create triggers on tables within **sysmaster**, but the database server never executes those triggers.)

Joins of multiple tables in **sysmaster** might return inconsistent results because the database server does not lock the tables during a join. You can join **sysmaster** tables with tables in other databases. However, to join **sysmaster** tables with tables in a nonlogging database, first make the nonlogging database the current database.

When you bring the database server up for the first time, it runs a script called **buildsmi**, which is in the **etc** directory. This script builds the database and tables that support SMI. The database server requires approximately 1750 free pages of logical-log space to build the **sysmaster** database.

If you receive an error message that directs you to run this script, a problem probably occurred while the database server was building the SMI database, tables, and views.

Warning: *When you use* ***buildsmi****, the existing* ***sysmaster*** *database is dropped and then re-created; if you have information in the ON-Archive catalog tables, it will be overwritten.*

Using the System-Monitoring Interface

This section describes the SMI tables and how you access them to monitor the database server operation.

What Are the SMI Tables?

The system-monitoring interface consists of tables and pseudo-tables that the database server maintains automatically. While the SMI tables appear to the user as tables, they are not recorded on disk like normal tables. Instead, the database server constructs the tables in memory, on demand, based on information in shared memory at that instant. When you query the SMI using one of these tables, the database server reads information from these shared-memory structures and returns it to you. Because the database server continually updates the data in shared memory, the information that SMI provides lets you examine the current state of your database server.

The SMI tables provide information on the following topics:

- Auditing
- Disk usage
- User profiling
- Database-logging status
- Tables
- Chunks
- Chunk I/O
- Dbspaces
- Locks

- Extents
- Virtual processor CPU usage
- System profiling

The data in the SMI tables changes dynamically as users access and modify databases that the database server manages.

Accessing SMI Tables

Any user can use SQL SELECT statements to query any SMI table except **sysadtinfo** and **sysaudit**. Only user **informix** can query **sysadtinfo** and **sysaudit**. Standard users cannot execute statements other than SELECT statements on SMI tables. Attempts to do so result in permission errors.

The user **informix** can execute SQL statements other than SELECT, but the results of such statements are unpredictable.

You cannot use **dbschema** or **dbexport** on any of the tables in the **sysmaster** database. If you do, the database server generates the following error message:

```
Database has pseudo tables - can't build schema
```

SELECT Statements

You can use SELECT statements on SMI tables wherever you can use SELECT against ordinary tables (from DB-Access, in a stored procedure, with ESQL/C, and so on) with one restriction: you cannot (meaningfully) reference **rowid** when you query SMI tables. SELECT statements that use **rowid** do not return an error, but the results are unpredictable.

All standard SQL syntax, including joins between tables, sorting of output, and so on, works with SMI tables. For example, if you want to join an SMI table with an non-SMI table, name the SMI table with the following standard syntax:

```
sysmaster:[@dbservername][owner.]tablename
```

Triggers and Event Alarms

Triggers based on changes to SMI tables never run. Although you can define triggers on SMI tables, triggers are activated only when an INSERT, UPDATE, or DELETE statement occurs on a table. The updates to the SMI data occur within the database server, without the use of SQL, so a trigger on an SMI table is never activated, even though the data returned by a SELECT statement indicates that it should be.

To create an event alarm, query for a particular condition at predefined intervals, and execute a stored procedure if the necessary conditions for the alarm are met.

SPL and SMI Tables

You can access SMI tables from within SPL. When you reference SMI tables, use the same syntax that you use to reference a standard table.

Locking and SMI Tables

The information in the SMI tables changes based on the database server activity. However, the database server does not update the information using SQL statements. When you use SMI tables with an isolation level that locks objects, it prevents other users from accessing the object, but it does not prevent the data from changing. In this sense, all the SMI tables have a permanent Dirty Read isolation level.

The System-Monitoring Interface Tables

The database server supports the following SMI tables.

Table	Description	Reference
sysadtinfo	Auditing configuration information	page 34-9
sysaudit	Auditing event masks	page 34-10
syschkio	Chunk I/O statistics	page 34-11
syschunks	Chunk information	page 34-11
sysconfig	Configuration information	page 34-13
sysdatabases	Database information	page 34-14
sysdbslocale	Locale information	page 34-15
sysdbspaces	Dbspace information	page 34-15
sysdri	Data-replication information	page 34-17
sysextents	Extent-allocation information	page 34-18
sysextspaces	External spaces information	page 34-18
syslocks	Active locks information	page 34-18
syslogs	Logical-log file information	page 34-19
sysprofile	System-profile information	page 34-20
sysptprof	Table information	page 34-23
syssesprof	Counts of various user actions	page 34-24
syssessions	Description of each user connected	page 34-26
sysseswts	User's wait time on each of several objects	page 34-28
systabnames	Database, owner, and table name for the tblspace	page 34-29
sysvpprof	User and system CPU used by each virtual processor	page 34-30

Many other tables in the **sysmaster** database are part of the system-monitoring interface but are not documented. Their schemas and column content can change from version to version.

sysadtinfo

The **sysadtinfo** table contains information about the auditing configuration for the database server. For more information, see your *Trusted Facility Manual*. You must be user **informix** or user **root** to retrieve information from the **sysadtinfo** table.

Column	Type	Description
adtmode	integer	If auditing is on or off ■ 0 off ■ 1 on
adterr	integer	Action on errors ■ 0 continually retry audit writes until they succeed. Processing for the thread that generated the error stops. ■ 1 write all failed audit writes to the message log and continue processing.
adtsize	integer	Maximum size of an audit file
adtpath	char(128)	Directory where audit files are written
adtfile	integer	Number of the audit file

sysaudit

For each defined audit mask (that is, for each *username*), the **sysaudit** table contains flags that represent the database events that generate audit records. The **success** and **failure** columns represent the bitmasks that compose the audit masks. If a bit is set in both the **success** the and **failure** columns, the corresponding event generates an audit record whether or not the event succeeded.

You must be user **informix** or **user** root to retrieve information from the **sysaudit** table.

Use the **onaudit** utility to list or modify an audit mask. For information about **onaudit**, and for additional information about auditing, see your *Trusted Facility Manual*.

Column	Type	Description
username	char(8)	Name of the mask
succ1	integer	Bitmask of the audit mask for success
succ2	integer	Bitmask of the audit mask for success
succ3	integer	Bitmask of the audit mask for success
succ4	integer	Bitmask of the audit mask for success
succ5	integer	Bitmask of the audit mask for success
fail1	integer	Bitmask of the audit mask for failure
fail2	integer	Bitmask of the audit mask for failure
fail3	integer	Bitmask of the audit mask for failure
fail4	integer	Bitmask of the audit mask for failure
fail5	integer	Bitmask of the audit mask for failure

syschkio

The **syschkio** table provides I/O statistics for individual chunks that the database server manages.

Column	Type	Description
chunknum	smallint	Chunk number
reads	integer	Number of physical reads
pagesread	integer	Number of pages read
writes	integer	Number of physical writes
pageswritten	integer	Number of pages written
mreads	integer	Number of physical reads (mirror)
mpagesread	integer	Number of pages read (mirror)
mwrites	integer	Number of physical writes (mirror)
mpageswritten	integer	Number of pages written (mirror)

syschunks

The **syschunks** table describes each of the chunks that the database server manages. In the **flags** and **mflags** columns, each bit position represents a separate flag. Thus, it might be easier to read values in the **flags** and **mflags** columns if the values are returned using the HEX function.

Column	Type	Description
chknum	smallint	Chunk number
dbsnum	smallint	Dbspace number
nxchknum	smallint	Number of the next chunk in this dbspace
chksize	integer	Number of pages in this chunk
offset	integer	Page offset of the chunk in its device or path

(1 of 3)

Column	Type	Description		
nfree	integer	Number of free pages in the chunk		
is_offline	integer	1 if the chunk is off-line, 0 if not		
is_recovering	integer	1 if the chunk is being recovered, 0 if not		
is_blobchunk	integer	1 if the chunk is in a blobspace, 0 if not		
is_inconsistent	integer	1 if the chunk is undergoing logical restore, 0 if not		
flags	smallint	Flags	Hexadecimal	Meaning
		16	0x0010	chunk is a mirrored chunk
		32	0x0020	chunk is in off-line mode
		64	0x0040	chunk is in on-line mode
		128	0x0080	chunk is in recovery mode
		256	0x0100	chunk has just been mirrored
		512	0x0200	chunk is part of a blobspace
		1024	0x0400	chunk is being dropped
		2048	0x0800	chunk is part of an optical stageblob
		4096	0x1000	chunk is inconsistent
		16384	0x4000	chunk contains temporary log space
		32768	0x8000	chunk was added during roll forward
fname	char(128)	Pathname for the device of this chunk		
mfname	char(128)	Pathname for the device of the mirrored chunk, if any		
moffset	integer	Page offset of the mirrored chunk		

(2 of 3)

Column	Type	Description
mis_offline	integer	1 if mirror is off-line, 0 if not
mis_recovering	integer	1 if mirror is being recovered, 0 if not
mflags	smallint	Mirrored chunk flags; values and meanings are the same as the **flags** column.

(3 of 3)

sysconfig

The **sysconfig** table describes the effective, original, and default values of the configuration parameters. For more information about the ONCONFIG file and the configuration parameters, see Chapter 33, "Configuration Parameters."

Column	Type	Description
cf_id	integer	Unique numeric identifier
cf_name	char(18)	Configuration parameter name
cf_flags	integer	Reserved for future use
cf_original	char(256)	Value in the ONCONFIG file at boot time
cf_effective	char(256)	Value effectively in use
cf_default	char(256)	Value provided by the database server if no value is specified in the ONCONFIG file

sysdatabases

The **sysdatabases** table describes each database that the database server manages.

Column	Type	Description
name	char(18)	Database name
partnum	integer	The partition number (tblspace identifier) for the **systables** table for the database
owner	char(8)	User ID of the creator of the database
created	date	Date created
is_logging	integer	1 if logging is active, 0 if not
is_buff_log	integer	1 if buffered logging, 0 if not
is_ansi	integer	1 if ANSI-compliant, 0 if not
is_nls	integer	1 if GLS-enabled, 0 if not
flags	smallint	Logging flags
		0 no logging
		1 unbuffered logging
		2 buffered logging
		4 ANSI-compliant database
		8 read-only database
		16 GLS database

sysdbslocale

The **sysdbslocale** table lists the locale of each database that the database server manages.

Column	Type	Description
dbs_dbsname	char(18)	Database name
dbs_collate	char(32)	The locale of the database

sysdbspaces

The **sysdbspaces** table describes each of the dbspaces that the database server manages. In the **flags** column, each bit position represents a separate flag. Thus, it might be easier to read values in the **flags** column if the values are returned using the HEX function.

Column	Type	Description		
dbsnum	smallint	Dbspace number		
name	char(18)	Dbspace name		
owner	char(8)	User ID of owner of the dbspace		
fchunk	smallint	Number of the first chunk in the dbspace		
nchunks	smallint	Number of chunks in the dbspace		
is_mirrored	integer	1 if dbspace is mirrored, 0 if not		
is_blobspace	integer	1 if the dbspace is a blobspace, 0 if not		
is_temp	integer	1 if the dbspace is a temporary dbspace, 0 if not		
flags	smallint	Flags	Hexadecimal	Meaning
		1	0x0001	dbspace has no mirror
		2	0x0002	dbspace uses mirroring
		4	0x0004	dbspace mirroring disabled

(1 of 2)

Column	Type	Description		
		8	0x0008	dbspace newly mirrored
		16	0x0010	blobspace
		32	0x0020	blobspace on removable media
		64	0x0040	blobspace on optical media
		128	0x0080	blobspace that has been dropped
		256	0x0100	blobspace is an optical stageblob
		512	0x0200	space is being recovered
		1024	0x0400	space has been physically recovered
		2048	0x0800	space is in logical recovery

(2 of 2)

sysdri

The **sysdri** table provides information on the high-availability data-replication status of the database server.

Column	Type	Description
type	char(50)	High-availability data replication type Possible values: primary secondary standard not initialized
state	char(50)	State of high-availability data replication Possible values: off on connecting failer read-only
name	char(20)	The name of the other database server in the high-availability data-replication pair
intvl	integer	The high-availability data-replication interval
timeout	integer	The high-availability data-replication timeout value for this database server
lostfound	char(128)	The pathname to the lost-and-found file

sysextents

The **sysextents** table provides information on extent allocation.

Column	Type	Description
dbsname	char(18)	Database name
tabname	char(18)	Table name
start	integer	Physical address for the extent
size	integer	Size of the extent, in pages

syslocks

The **syslocks** table provides information on all the currently active locks in the database server.

Column	Type	Description	
dbsname	char(18)	Database name	
tabname	char(18)	Table name	
rowidlk	integer	Real rowid, if it is an index key lock	
keynum	smallint	Key number of index key lock	
type	char(4)	Type of lock	
		B	byte lock
		IS	intent shared lock
		S	shared lock
		XS	shared key value held by a repeatable reader
		U	update lock
		IX	intent exclusive lock
		SIX	shared intent exclusive lock

(1 of 2)

Column	Type	Description
		X exclusive lock
		XR exclusive key value held by a repeatable reader
owner	integer	Session ID of the lock owner
waiter	integer	Session ID of the user waiting for the lock. If more than one user is waiting, only the first session ID appears.

(2 of 2)

syslogs

The **syslogs** table provides information on space use in logical-log files. It has the following columns. In the **flags** column, each bit position represents a separate flag. For example, for a log file, the **flags** column can have flags set for both current log file and temporary log file. Thus, it might be easier to read values in the **flags** column if the values are returned using the HEX function.

Column	Type	Description
number	smallint	Logical-log file number
uniqid	integer	Log-file ID
size	integer	Number of pages in the log file
used	integer	Number of pages used in the log file
is_used	integer	1 if file is used, 0 if not
is_current	integer	1 if file is the current file, 0 if not
is_backed_up	integer	1 if file has been backed up, 0 if not
is_new	integer	1 if the log has been added since the last level-0 dbspace backup, 0 if not
is_archived	integer	1 if file has been placed on the archive tape, 0 if not
is_temp	integer	1 if the file is flagged as a temporary log file, 0 if not

(1 of 2)

Column	Type	Description		
flags	smallint	Flags	Hexadecimal	Meaning
		1	0x01	log file in use
		2	0x02	current log file
		4	0x04	log file has been backed up
		8	0x08	newly added log file
		16	0x10	log file has been written to dbspace backup tape
		32	0x20	log is a temporary log file

(2 of 2)

sysprofile

The **sysprofile** table contains profile information about the database server.

Column	Type	Description
name	char(13)	Name of profiled event (see table that follows for a list of possible events)
srtspmax	integer	Maximum disk space required by a sort
totsorts	integer	Total number of sorts performed
value	integer	Value of profiled event (see table that follows for a list of possible events)

The following table lists the events that, together with a corresponding value, make up the rows of the **sysprofile** table.

Events Profiled in sysprofile	Description
dskreads	Number of actual reads from disk
bufreads	Number of reads from shared memory.
dskwrites	Actual number of writes to disk
bufwrites	Number of writes to shared memory
isamtot	Total number of calls
isopens	isopen calls
isstarts	isstart calls
isreads	isread calls
iswrites	iswrite calls
isrewrites	isrewrite calls
isdeletes	isdelete calls
iscommits	iscommit calls
isrollbacks	isrollback calls
ovlock	Overflow lock table
ovuser	Overflow user table
ovtrans	Overflow transaction table
latchwts	Latch request waits
bufwts	Buffer waits
lockreqs	Lock requests
lockwts	Lock waits
ckptwts	Checkpoint waits

(1 of 3)

Events Profiled in sysprofile	Description
deadlks	Deadlocks
lktouts	Deadlock time-outs
numckpts	Number checkpoints
plgpagewrites	Physical-log pages written
plgwrites	Physical-log writes
llgrecs	Logical-log records
llgpagewrites	Logical-log writes
llgwrites	Logical-log pages written
pagreads	Page reads
pagwrites	Page writes
flushes	Buffer-pool flushes
compress	Page compresses
fgwrites	Foreground writes
lruwrites	Least-recently used (LRU) writes
chunkwrites	Writes during a checkpoint
btradata	Data pages read ahead through leaf
btraidx	Leaf read aheads through parent
dpra	Data-page read aheads
rapgs_used	Read-ahead pages used
seqscans	Sequential scans
totalsorts	Total sorts

(2 of 3)

Events Profiled in sysprofile	Description
memsorts	Sorts that fit in memory
disksorts	Sorts that did not fit in memory
maxsortspace	Maximum disk space used by a sort

(3 of 3)

sysptprof

The **sysptprof** table lists information about a tblspace, also referred to as a partition. Tblspaces corresponds to tables.

Profile information for a table is available only when a table is open. When the last user who has a table open closes it, the partition structure in shared memory is freed and thus any profile statistics are lost.

Column	Type	Description
dbsname	char(18)	Database name
tabname	char(18)	Table name
partnum	integer	Partition (tblspace) number
lockreqs	integer	Number of lock requests
lockwts	integer	Number of lock waits
deadlks	integer	Number of deadlocks
lktouts	integer	Number of lock timeouts
isreads	integer	Number of isreads
iswrites	integer	Number of iswrites
isrewrites	integer	Number of isrewrites
isdeletes	integer	Number of isdeletes
bufreads	integer	Number of buffer reads

(1 of 2)

Column	Type	Description
bufwrites	integer	Number of buffer writes
seqscans	integer	Number of sequential scans
pagreads	integer	Number of page reads
pagwrites	integer	Number of page writes

(2 of 2)

syssesprof

The **syssesprof** table lists cumulative counts of the number of occurrences of user actions such as writes, deletes, or commits.

Column	Type	Description
sid	integer	Session ID
lockreqs	integer	Number of locks requested
locksheld	integer	Number of locks currently held
lockwts	integer	Number of times waited for a lock
deadlks	integer	Number of deadlocks detected
lktouts	smallint	Number of deadlock timeouts
logrecs	integer	Number of logical-log records written
isreads	integer	Number of reads
iswrites	integer	Number of writes
isrewrites	integer	Number of rewrites
isdeletes	integer	Number of deletes
iscommits	integer	Number of commits
isrollbacks	integer	Number of rollbacks
longtxs	integer	Number of long transactions

(1 of 2)

Column	Type	Description
bufreads	integer	Number of buffer reads
bufwrites	integer	Number of buffer writes
seqscans	integer	Number of sequential scans
pagreads	integer	Number of page reads
pagwrites	integer	Number of page writes
total_sorts	integer	Number of total sorts
dsksorts	integer	Number of sorts that did not fit in memory
max_sortdiskspace	integer	Number of maximum space used by a sort
logspused	integer	Number of bytes of logical-log space used by current transaction of session
maxlogsp	integer	Maximum number of bytes of logical-log space ever used by the session

(2 of 2)

syssessions

The **syssessions** table provides general information on each user connected to the database server. In the **state** column, each bit position represents a separate flag. Thus, it might be easier to read values in the **state** column if the values are returned using the HEX function.

Column	Type	Description
sid	integer	Session ID
username	char(8)	User ID
uid	smallint	User ID number
pid	integer	Process ID of the client
hostname	char(16)	Hostname of client
tty	char(16)	Name of the user's **stderr** file
connected	integer	Time that user connected to the database server
feprogram	char(16)	Reserved for future use
pooladdr	integer	Session pool address
is_wlatch	integer	1 if the primary thread for the session is waiting on a latch
is_wlock	integer	1 if the primary thread for the session is waiting on a lock
is_wbuff	integer	1 if the primary thread for the session is waiting on a buffer
is_wckpt	integer	1 if the primary thread for the session is waiting on a checkpoint
is_wlogbuf	integer	1 if the primary thread for the session is waiting on a log buffer
is_wtrans	integer	1 if the primary thread for the session is waiting on a transaction
is_monitor	integer	1 if the session is a special monitoring process
is_incrit	integer	1 if the primary thread for the session is in a critical section

(1 of 2)

Column	Type	Description		
state	integer	Flags	Hexadecimal	Meaning
		1	0x00000001	user structure in use
		2	0x00000002	waiting for a latch
		4	0x00000004	waiting for a clock
		8	0x00000008	waiting for a buffer
		16	0x00000010	waiting for a checkpoint
		32	0x00000020	in a read RSAM call
		64	0x00000040	writing logical-log file to backup tape
		128	0x00000080	ON-Monitor (UNIX-only)
		256	0x00000100	in a critical section
		512	0x00000200	special daemon
		1024	0x00000400	archiving
		2048	0x00000800	clean up dead processes
		4096	0x00001000	waiting for write of log buffer
		8192	0x00002000	special buffer-flushing thread
		16384	0x00004000	remote database server
		32768	0x00008000	deadlock timeout used to set RS_timeout
		65536	0x00010000	regular lock timeout
		262144	0x00040000	waiting for a transaction
		524288	0x00080000	primary thread for a session
		1048576	0x00100000	thread for building indexes
		2097152	0x00200000	btree cleaner thread

(2 of 2)

sysseswts

The **sysseswts** table provides information on the amount of time that users wait for various database objects.

Column	Type	Description
sid	integer	Session ID
reason	char(16)	Description of reason for wait: ■ unspecified ■ buffer ■ lock ■ asynchronous I/O ■ mt yield 0 ■ mt yield n ■ mt yield ■ checkpoint ■ log i/o ■ log copy ■ condition ■ lock mutex ■ lockfree mutex ■ deadlock mutex ■ lrus mutex ■ tblsp mutex ■ log mutex ■ ckpt mutex ■ mutex ■ mt ready ■ mt yield x ■ running

(1 of 2)

Column	Type	Description
numwaits	integer	Number of waits for this reason
cumtime	float	Cumulative time waited for this reason in microseconds
maxtime	integer	Maximum time waited during this session for this reason

(2 of 2)

systabnames

The **systabnames** table describes each table that the database server manages.

Column	Type	Description
partnum	integer	Tblspace identifier
dbsname	char(18)	Database name
owner	char(8)	User ID of owner
tabname	char(18)	Table name
collate	char(32)	Collation associated with a GLS database

sysvpprof

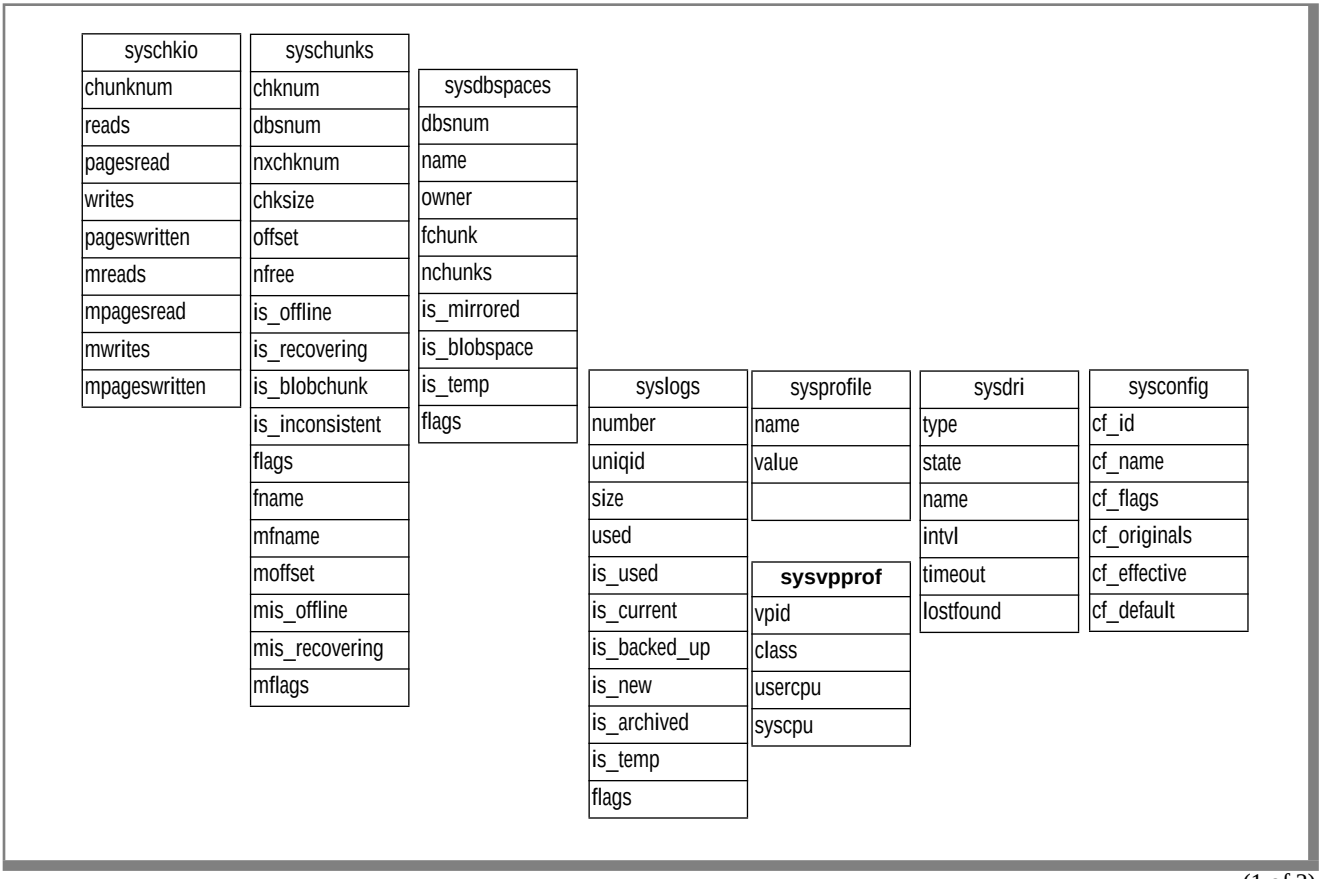
The **sysvpprof** table lists user and system CPU time for each virtual processor.

Column	Type	Description
vpid	integer	Virtual processor ID
class	char(50)	Type of virtual processor: ■ cpu ■ adm ■ lio ■ pio ■ aio ■ tli ■ soc ■ str ■ shm ■ opt ■ msc ■ adt
usercpu	float	Number of microseconds of user time
syscpu	float	Number of microseconds of system time

The SMI Tables Map

Figure 34-1 displays the columns in the SMI tables.

Figure 34-1
Columns in the SMI tables



(1 of 2)

sysseswts
sid
reason
numwaits
cumtime
maxtime

syssesprof
sid
lockreqs
locksheld
lockwts
deadlks
lktouts
logrecs
isreads
iswrites
isrewrites
isdeletes
iscommits
isrollbacks
longtxs
bufreads
bufwrites
seqscans
pagreads
pagwrites
total_sorts
dsksorts
max_sort diskspace
logspused
maxlogsp

syssessions
sid
username
uid
pid
hostname
tty
connected
feprogram
pooladdr
is_wlatch
is_wlock
is_wbuff
is_wckpt
is_wlogbuf
is_wtrans
is_monitor
is_incrit
state

syslocks
dbsname
tabname
rowidlk
keynum
type
owner (sid)
waiter (sid)

sysextents
dbsname
tabname
start
size

sysptprof
dbsname
tabname
partnum
lockreqs
lockwts
deadlks
lktouts
isreads
iswrites
isrewrites
isdeletes
bufreads
bufwrites
seqscans
pagreads
pagwrites

systabnames
partnum
dbsname
owner
tabname
collate

sysdatabases
name
partnum
owner
created
is_logging
is_buff_log
is_ansi
is_nls
flags

(2 of 2)

Using SMI Tables to Obtain onstat Information

You can obtain information provided by the **onstat** utility by querying appropriate SMI tables using SQL. The following table indicates which SMI tables to query to obtain the information provided by a given **onstat** option. For descriptions of the **onstat** options, see "onstat: Monitor Database Server Operation" on page 35-62.

onstat Option	SMI Tables to Query	onstat Fields Not in SMI Tables
-d	sysdbspaces syschunks	address bpages
-D	sysdbspaces syschkio	
-F	sysprofile	address flusher snoozer state data
-g dri	sysdri	Last DR CKPT (id / pg) DRAUTO
-g glo	sysvpprof	listing of virtual processors by class
-k	syslocks	address lklist tblsnum

(1 of 2)

onstat Option	SMI Tables to Query	onstat Fields Not in SMI Tables
-l	syslogs sysprofile	all physical-log fields (except numpages and numwrits) all logical-log buffer fields (except numrecs, numpages, and numwrits) address begin % used
-p	sysprofile	
-u	syssessions syssesprof	address wait nreads nwrites

(2 of 2)

Chapter

35

Utilities

The -V Option 35-4

Multibyte Characters 35-4

oncheck: Check, Repair, or Display 35-5

ondblog: Change Logging Mode 35-18

oninit: Initialize Dynamic Server 35-20

onlog: Display Logical-Log Contents. 35-23

onmode: Mode and Shared-Memory Changes 35-28

onparams: Modify Log-Configuration Parameters 35-43

onspaces: Manage Database Spaces 35-48

onstat: Monitor Database Server Operation 35-62

ontape: Logging, Archives, and Restore. 35-95

This chapter consists of reference material for the Dynamic Server utilities. These utilities let you perform administrative tasks from the command line. This chapter documents the following utilities:

- **oncheck**
- **ondblog**
- **oninit**
- **onlog**
- **onmode**
- **onparams**
- **onspaces**
- **onstat**
- **ontape**

Except in the case of **oninit** and some **oncheck** and **onlog** options, you must initialize the database server before you execute any of the utilities.

The -V Option

All of the Informix command-line utilities let you use the **-V** option. This option displays the software version number and the serial number. You use the **-V** option primarily for debugging. When an Informix Technical Support representative asks for the version number, you can use **-V** to find the information.

GLS

Multibyte Characters

The database server utilities support multibyte command-line arguments. For a complete listing of the utilities that support multibyte command-line arguments, see the *Informix Guide to GLS Functionality*.

oncheck: Check, Repair, or Display

Depending on the options that you choose, **oncheck** can perform the following functions:

- Check specified disk structures for inconsistencies
- Repair indexes that are found to contain inconsistencies
- Display information about the disk structures

For background information on the various disk structures that the database server manages, see Chapter 38, "Disk Structures and Storage."

The **oncheck** utility is frequently confused with **onstat**. For a comparison of **oncheck** and **onstat**, see "Monitoring with onstat and oncheck Utilities" on page 29-12.

oncheck Check-and-Repair Options

The **oncheck** utility can repair the following types of structures:

- Partition page statistics
- Bitmap pages
- Partition blobpages
- Blobspace blobpages
- Indexes

If **oncheck** detects inconsistencies in other structures, messages alert you to these inconsistencies, but **oncheck** cannot resolve the problem. For more details about database server consistency checking, see Chapter 27, "What Is Consistency Checking?"

Use the **-y** option to instruct **oncheck** to perform repair actions automatically. For example, the following commands instruct **oncheck** to perform repairs automatically:

```
oncheck -cd -y
oncheck -cD -y
oncheck -ci -y
oncheck -cI -y
```

If you do not use the **-y** option, **oncheck** prompts you when it encounters an inconsistency and allows you to request a repair. If you specify option **-n**, **oncheck** does not prompt you because this option instructs **oncheck** to not perform repairs.

The **oncheck** utility does not repair fragmented indexes. If **oncheck** encounters an inconsistency in a fragmented index, it displays a message similar to the following:

```
Please Drop and Re-create Index index_name for table_name.
```

Any user can execute the check options. Only user **informix** or **root** can display database data or initiate repair options.

What Does Each Option Do?

As shown in Figure 35-1 on page 35-7, the **oncheck** options are divided into three categories: check, repair, and display. The display or print options (those prefixed with the letter p) are identical in function to the **-c** options, except that the **-p** options display additional information about the data that is being checked as the **oncheck** utility executes.

In general, the **-c** options check for consistency and display a message on your screen only if they find an error or inconsistency.

Figure 35-1 associates **oncheck** options with their function.

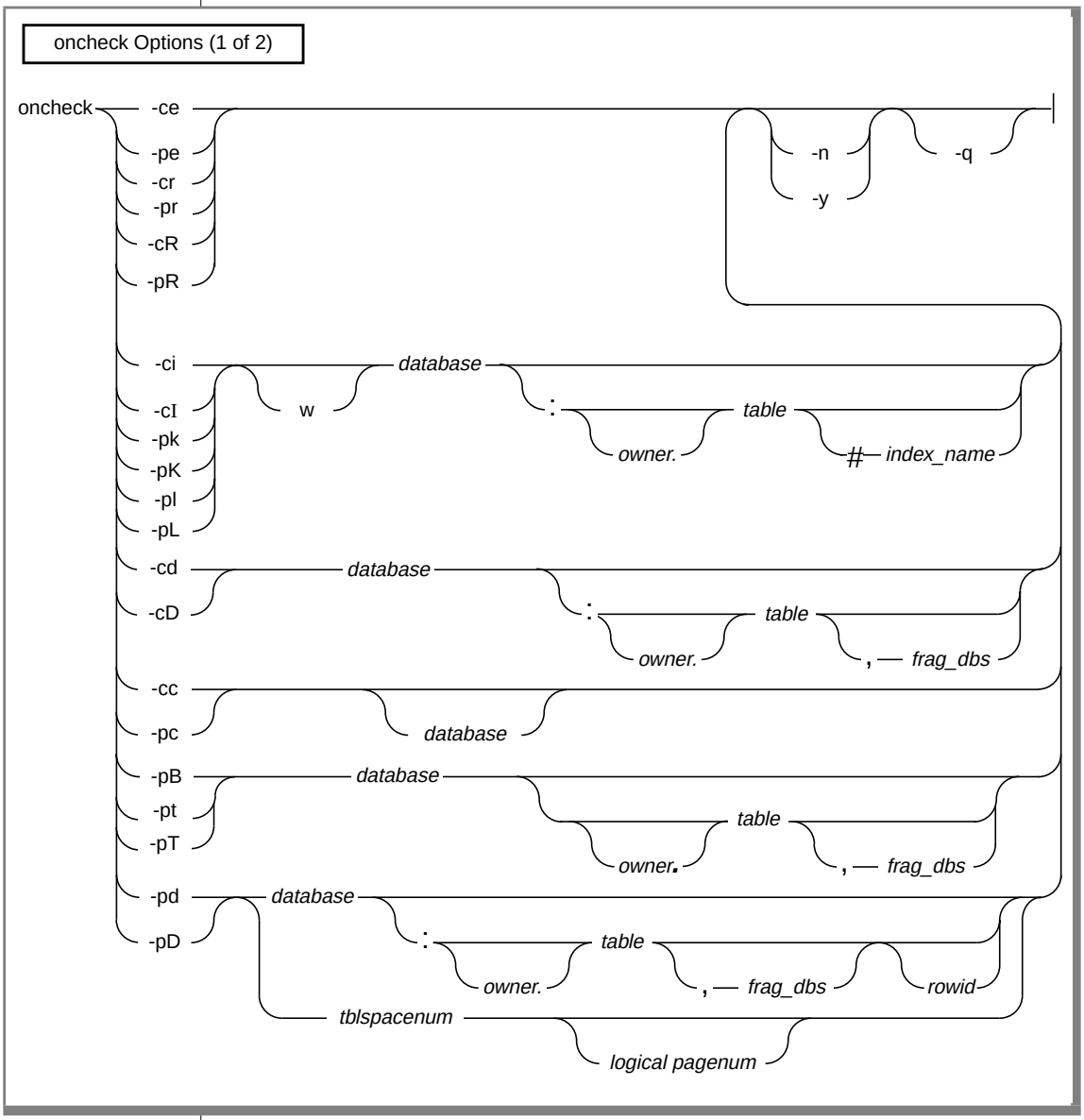
Figure 35-1
oncheck Options and Their Function

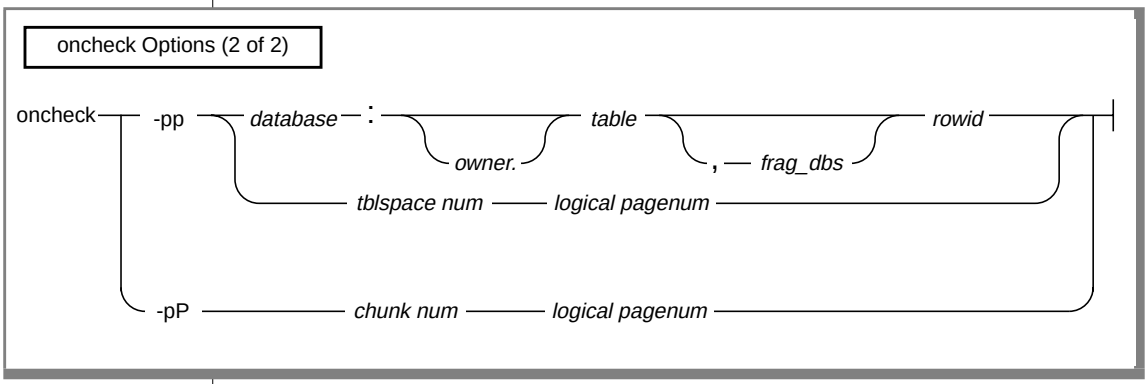
Object	Check	Repair	Display
Blobspace TEXT and BYTE values			-pB
Chunks and extents	-ce		-pe
Data rows, no TEXT and BYTE values	-cd		-pd
Data rows, TEXT and BYTE values	-cD		-pD
Index (key values)	-ci, -ciw	-ci -y -pk -y, -pkw -y	-pk
Index (keys plus rowids)	-cI, -cIw	-cI -y -pK -y, -pKw -y	-pK
Index (leaf key values)		-pl -y, -plw -y	-pl
Index (leaf keys plus rowids)		-pL -y, -pLw -y	-pL
Pages (by table or fragment)			-pp
Pages (by chunk)			-pP
Root reserved pages	-cr		-pr
Space usage (by table or fragment)			-pt
Space usage (by table, with indexes)			-pT
System catalog tables	-cc		-pc

Locking and oncheck

The **oncheck** utility places a shared lock on tables when it checks data and indexes. It also places a shared lock on system catalog tables when they are checked. It places an exclusive lock on a table when it executes repair options.

Syntax


oncheck Options (1 of 2)
oncheck
-ce
-pe
-cr
-pr
-cR
-pR
-n
-y
-q
-ci
-cI
-pk
-pK
-pl
-pL
w
database
:
owner.
table
#
index_name
-cd
-cD
database
:
owner.
table
,
frag_dbs
-cc
-pc
database
-pB
-pt
-pT
database
owner.
table
,
frag_dbs
-pd
-pD
database
:
owner.
table
,
frag_dbs
rowid
tblspacenum
logical pagenum



Option Descriptions

You cannot combine **oncheck** option flags except as described in the paragraphs that follow.

Using the -cc Option to Check System Catalog Tables

The **-cc** option checks each of the system catalog tables for the specified database. If the database is omitted, all system catalog tables for all databases are checked. Before you execute **oncheck**, execute the SQL statement UPDATE STATISTICS to ensure that an accurate check occurs.

To check a table, **oncheck** compares each system catalog table to its corresponding entry in the tblspace tblspace. (See "Structure of the Tblspace Tblspace" on page 38-19.) The **-pc** option performs the same checks and also displays the system catalog information as it checks it, including extent use for each table.

```
% oncheck -cc
% oncheck -cc stores7
```

Using the -cd and -cD Options to Check Pages

The **-cd** option reads all pages, excluding blobpages, from the tblspace for the specified database, table, or fragment and checks each page for consistency. Entries in the bit-map page (see "Structure of a Dbspace Bit-Map Page" on page 38-25) are checked against the pages to verify mapping.

If the database contains fragmented tables, and you do not specify a fragment, all fragments in the table are checked. If a table is not specified, all tables in the database are checked. (The **-pd** option displays a hexadecimal dump of specified pages but does not check for consistency.)

The **-cD** option performs checks that are similar to the checks performed when you use the **-cd option,** but it includes a consistency check of dbspace blobpages if any exist. The **-cD** option does not read an entire blobpage as it does with standard pages. Instead, **oncheck** reads only the header of each blobpage and checks it for consistency. This limited consistency checking is done to avoid the performance degradation that could occur if **oncheck** were to read each blobpage. Because **oncheck** does not read the entire page, it does not compare beginning time stamps (stored in the header) with ending time stamps (stored at the end of a blobpage).

To monitor blobspace blobpages, refer to **oncheck -pB** ("Using the -pB Option to Display Blobspace Statistics" on page 35-13).

```
% oncheck -pB stores7:catalog
```

If **oncheck** finds an inconsistency, a message similar to the following example is displayed:

```
BAD PAGE 20001c:pg_addr 20001c != bp-> bf_pagenum 200045
```

If no inconsistencies are found, **oncheck** displays a header similar to the one in the following example for each table that it checks:

```
TBLSPACE data check for stores7:informix.customer
```

The **oncheck** utility displays a header similar to the following example for fragmented tables, one per fragment:

```
TBLspace data check for stores7:informix.tab1
 Table fragment in DBspace db1
```

If you specify a single fragment, **oncheck** displays a single header for that fragment.

The **oncheck** utility locks each table as its indexes are checked for both the **-cd** and **-cD** options.

Using the -ce Option to Check the Chunk Free List

The **-ce** option checks each chunk free list and corresponding free space and each tblspace extent. (See "Next-Extent Allocation" on page 38-31 and "Structure of the Chunk Free-List Page" on page 38-18, respectively.) The **oncheck** process verifies that the extents on disk correspond to the current control information describing them.

The **-pe** option performs the same checks and also displays the chunk and tblspace extent information as it checks it.

```
% oncheck -ce
```

Using the -ci and -cI Options to Check Index Node Links

The **-ci** option checks the ordering of key values and the consistency of horizontal and vertical node links for all indexes associated with the specified table. (See "Structure of B-Tree Index Pages" on page 38-48.)

If an index is not specified, all indexes are checked.

If a table is not specified, all tables in the database are checked.

If inconsistencies are detected, you are prompted for confirmation to repair the problem index. If you specify the **-y** (yes) option, indexes are automatically repaired. If you specify the **-n** (no) option, the problem is reported but not repaired. No prompting occurs.

If **oncheck** does not find inconsistencies, the only message displayed is as follows:

```
validating indexes......
```

The message is followed by the names of the indexes that **oncheck** is checking.

Index rebuilding can be time consuming if you use **oncheck**. Processing is usually faster if you use the DROP INDEX and CREATE INDEX SQL statements to drop the index and re-create it.

The **-cI** option performs the same checks as **-ci**, but it also checks that the key value tied to a rowid in an index is the same as the key value in the row. The same **-ci** repair options are available with **-cI**.

The following example checks all indexes on the **customer** table:

```
% oncheck -cI -n stores7:customer
```

The following example checks the index **zip_ix** on the **customer** table:

```
% oncheck -cI -n stores7:customer#zip_ix
```

By default, both the **-ci** and **-cI** options cause **oncheck** to lock each table as it checks the indexes that belong to the table. You can append option **w**, however, to instruct **oncheck** not to place a shared lock on a table. However, if you use option **w** together with the repair option, **-y**, **oncheck** still places an exclusive lock on the table when it must perform a repair action. For more information on option **w**, see "Using Option w to Turn Off Locking" on page 35-17.

Using the -cr and -cR Options to Check Reserved Pages

The **-cr** option checks each of the root dbspace reserved pages (see "Reserved Pages" on page 38-6) as follows:

- It validates the contents of the ONCONFIG file with the PAGE_CONFIG reserved page.
- It ensures that all chunks can be opened, that chunks do not overlap, and that chunk sizes are correct.

The following example checks each of the root dbspace reserved pages:

```
% oncheck -cr
```

The **-cR** option performs the same operations as the **-cr** option, but it also checks all logical-log and physical-log pages for consistency. The **-cr** option is considerably faster because does not check the log file pages.

If you have changed the value of a configuration parameter (either through ON-Monitor or by editing the configuration file), and you have not yet reinitialized shared memory, **oncheck -cr** and **oncheck -cR** detects the inconsistency and returns an error message.

The **-pr** and **-pR** options perform the same checks as **oncheck -cr** and **oncheck -cR**, respectively, and also displays the reserved-page information as it checks the reserved pages.

If **oncheck -cr** does not display any error messages after you execute it, you can assume that all three items in the preceding list were checked successfully.

Using the -pB Option to Display Blobspace Statistics

The **-pB** option displays statistics that describe the average fullness of blobspace blobpages in a specified table. These statistics provide a measure of storage efficiency for individual TEXT and BYTE objects in a database or table. If a table or fragment is not specified, statistics are displayed for the entire database. (See "Optimizing Blobspace Blobpage Size" on page 14-21.)

```
% oncheck -pB photo_base:photos
```

Using the -pd and -pD Options to Display Rows in Hexadecimal Format

The **-pd** option takes a database, a table, a fragment, and a specific rowid or tblspace number and logical page number as input. In every case, **-pd** prints page-header information and displays the specified rows for the database object (database, table, fragment, rowid, or page number) that you specify in hexadecimal and ASCII format. No checks for consistency are performed.

If you specify a rowid (expressed as a hexadecimal value), the rowid maps to a particular page, and all rows from that page are printed.

If you specify a logical page number (expressed as a decimal), all the rows of the tblspace number with the logical page number are printed.

If you specify a fragment, all the rows in the fragment are printed, with their rowids, forward pointers, and page type.

If you specify a table, all the rows in the table are printed, with their rowids, forward pointers, and page type.

If you specify a database, all the rows in all the tables in the database are printed. TEXT and BYTE column descriptors stored in the data row are printed, but TEXT and BYTE data itself is not.

The **-pD** option prints the same information as **-pd**. In addition, **-pD** prints TEXT and BYTE values stored in the tblspace or header information for TEXT and BYTE objects stored in a blobspace blobpage.

```
% oncheck -pd stores7:customer,frgmnt1
% oncheck -pd stores7:customer
% oncheck -pD stores7:customer 0x101
```

Using the -pk and -pK, -pl, and -pL Options to Display Index Information

Repair options are available for each option.

The **-pk** option performs the same checks as the **-ci** option. (See "Using the -ci and -cI Options to Check Index Node Links" on page 35-11.) In addition, **-pk** displays the key values for all indexes on the specified table as it checks them.

The **-pK** option performs the same checks as the **-cI** option. (See "Using the -ci and -cI Options to Check Index Node Links" on page 35-11.) The **-pK** option displays the key values and rowids as it checks them.

The **-pl** option performs the same checks as the **-ci** option and displays the key values, but only leaf-node index pages are checked. The root and branch-node pages are ignored. See "Structure of B-Tree Index Pages" on page 38-48.

The **-pL** option performs the same checks as the **-cI** option and displays the key values and rowids, but only for leaf-node index pages. The root and branch-node pages are ignored.

```
% oncheck -pL -n stores7.customer
```

The following example displays information on all indexes on the **customer** table:

```
% oncheck -pl -n stores7:customer
```

The following example displays information about the index **zip_ix,** which was created on the **customer** table:

```
% oncheck -pl -n stores7:customer#zip_ix
```

Using the -pp and -pP Options to Display the Contents of a Logical Page

The **-pp** option requires as input either of the following values:

- A table name and a rowid

  If the table that you want to check is fragmented, you must also supply the name of the dbspace in which the fragment is located.
- A tblspace number and logical page number

Use the **-pp** option to dump the contents of the logical page number contained in the rowid. The page contents appear in ASCII format. The display also includes the number of slot-table entries on the page.

To obtain the rowid of a specific data row, use **oncheck -pD**.

The **-pP** option provides the same information as the **-pp** option but requires a chunk number and logical page number as input.

```
% oncheck -pp stores7:customer,frag_dbspce1 0x211
% oncheck -pp stores7:orders 0x211
% oncheck -pP 0x100000a 25
% oncheck -pP 3 15
```

Using -pr Option to Display Reserved-Page Information

The **-pr** option performs the same checks as the **-cr** option. (For a description of the **-cr** option, see "Using the -cr and -cR Options to Check Reserved Pages" on page 35-12.) In addition, **-pr** displays the reserved-page information as it checks the reserved pages. For a listing and explanation of **oncheck -pr** output, see "Reserved Pages" on page 38-6.

```
% oncheck -pr
```

If you have changed the value of a configuration parameter (either through ON-Monitor or by editing the ONCONFIG file), and you have not yet reinitialized shared memory, **oncheck -cr** detects the inconsistency, prints both values, and displays an error message.

Using the -pt and -pT Options to Display Tblspace Information for a Particular Table or Fragment

The **-pt** option prints a tblspace report for a given table or fragment whose name and database you specify when you execute **oncheck** at the command line. The report contains general allocation information including the maximum row size, the number of keys, the number of extents, their sizes, the pages allocated and used per extent, the current serial value, and the date that the table was created. The Extents fields list the physical address for the tblspace tblspace entry for the table and the address of the first page of the first extent. If a table is not specified, this information is displayed for all tables in the database.

The **-pT** option prints the same information as the **-pt** option. In addition, the **-pT** option displays index-specific information and page-allocation information by page type (for dbspaces).

Output for both **-pt** and **-pT** contains listings for `Number of pages used`. The value shown in the output for this field is never decremented because the disk space allocated to a tblspace as part of an extent remains dedicated to that extent even after space is freed by deleting rows. (See your *Performance Guide*.) For an accurate count of the number of pages currently used, refer to the detailed information on tblspace use (organized by page type) that the **-pT** option provides.

```
% oncheck -pT stores7:customer
```

Using Option w to Turn Off Locking

If you append option **w** to the index-checking options, **oncheck** will not place a shared lock on affected tables while checking the indexes. Instead the **w** option places an IS (intent shared) lock on the table, which prevents actions such as dropping the table or the indexes during the check.

Using option **w** allows users to manipulate table rows, performing inserts, updates, and deletions, while **oncheck** checks or prints the indexes. You can append option **w** to the **-ci**, **-cI**, **-pk**, **-pK**, **-pl**, and **-pL** options. For example, the following sample command instructs **oncheck** not to lock indexes for the **customer** table while it validates the order of key values, validates horizontal links, and ensures that no node appears twice in the index:

```
oncheck -ciw stores7:customer
```

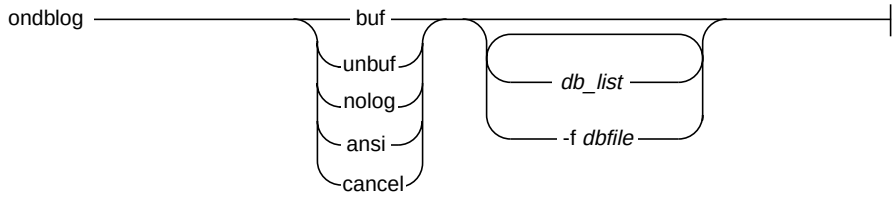
When you specify option **w**, **oncheck** does not lock indexes for tables that use row lock mode. If **oncheck** detects page lock mode, it displays a warning message and places a shared lock on the table regardless.

ondblog: Change Logging Mode

The **ondblog** utility lets you change the logging mode for one or more databases.

If you add logging to a database, you must create a level-0 backup before the change takes effect.

Syntax



Element	Purpose	Key Considerations
buf	Sets the logging mode so that transaction information is written to a buffer before it is written to a logical log.	None.
unbuf	Sets the logging so that data is not written to a buffer before it is written to a logical log.	None.
nolog	Sets the logging so that no database transactions are logged.	None.
ansi	Change the logging of the database to be ANSI compliant.	None.
cancel	Cancels the logging mode change request before the next level-0 backup occurs.	None.
-f *dbfile*	Changes the logging status of the databases that are listed (one per line) in the text file whose pathname is given by *dbfile*.	**Additional Information:** This command is useful if the list of databases is very long or often used.
db_list	Names a space-delimited list of databases whose logging status is to be changed.	**Additional Information:** If you do not specify anything, all databases that are managed by the database server are modified.

UNIX

On UNIX, the Database option in the ON-Monitor **Logical-Logs** menu is similar to these **ondblog** command options. ♦

For information on how to change the logmode using ON-Archive, see your *Archive and Backup Guide*. For information about changing the logging status of a database, see Chapter 17, "Managing Database-Logging Status."

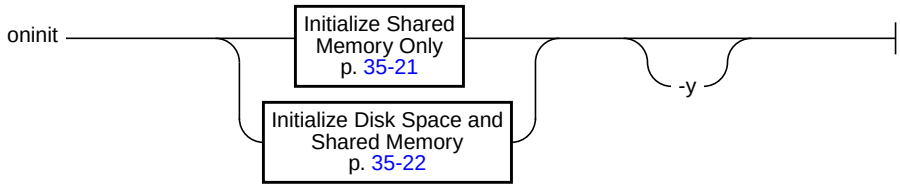
oninit: Initialize Dynamic Server

The **oninit** utility initializes database server shared memory and brings the database server on-line.

If you use **oninit** options, you can also initialize disk space. You must be logged in as **root** or user **informix** to execute **oninit**.

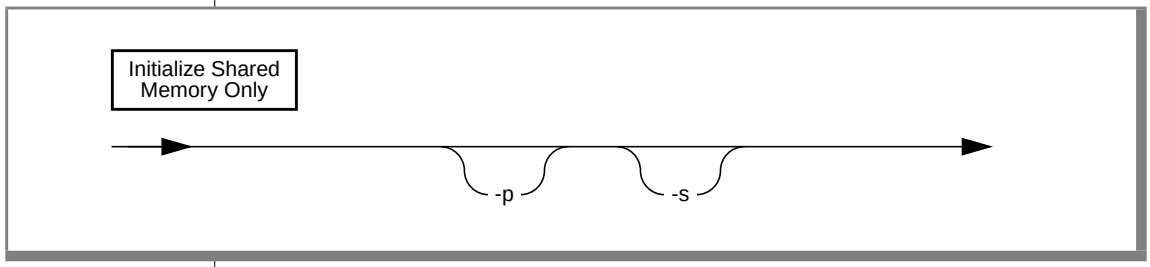
For information about what happens during initialization, see Chapter 8, "Initializing the Database Server."

Syntax



Element	Purpose	Key Considerations
-y	Causes the database server to automatically respond *yes* to all prompts.	None.

Initialize Shared Memory Only



Element	Purpose	Key Considerations
-p	Directs **oninit** not to search for (and delete) temporary tables.	**Additional Information**: If you use this option, the database server returns to on-line mode more rapidly, but space used by temporary tables left on disk is not reclaimed.
-s	Initializes shared memory and leaves the database server in quiescent mode. See "Initializing Shared Memory with the -s Option" on page 35-21.	**Additional Information**: This option is equivalent to the ON-Monitor Mode menu, Startup option. The database server should be in off-line mode to initialize shared memory.

Initializing Shared Memory with No Options

If you execute **oninit** without options, the database server is left in on-line mode after shared memory is initialized. For example, the following commands take the database server off-line and then reinitialize shared memory:

```
% onmode -ky
% oninit
```

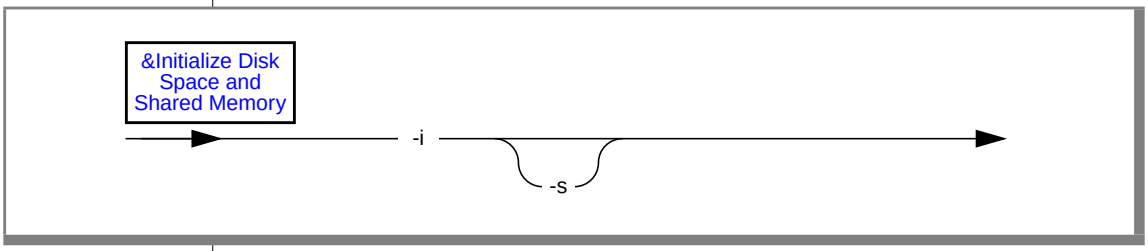
Initializing Shared Memory with the -s Option

The **-s** option initializes shared memory and leaves the database server in quiescent mode. This option is equivalent to the ON-Monitor Mode menu, Startup option.

The following commands take the database server off-line, then reinitialize shared memory, and leave the database server in quiescent mode:

```
% onmode -ky
% oninit -s
```

&Initialize Disk Space and Shared Memory



Element	Purpose	Key Considerations
-i	Causes the database server to initialize disk space and shared memory. Leaves the database server in on-line mode after initializing disk space.	None.
-s	When used with **-i**, the **-s** option causes the database server to be left in quiescent mode after disk initialization.	None.

Warning: *When you initialize disk space, the initialization destroys all data that your database server currently manages.*

The database server must be off-line when you initialize disk space.

Specify the Number of Virtual Processors

Use NUMCPUVPS and NUMAIOVPS to specify the initial number of VPs for the CPU and AIO classes. For more information about these parameters, see "NUMCPUVPS" on page 33-63 and "NUMAIOVPS" on page 33-62.

onlog: Display Logical-Log Contents

The **onlog** utility displays the contents of a logical-log file, either on disk or on a backup tape created by **ontape**. (To display the contents of backup tapes created by ON-Archive, use the LIST / LOGRECORDS command as described in your *Archive and Backup Guide*.)

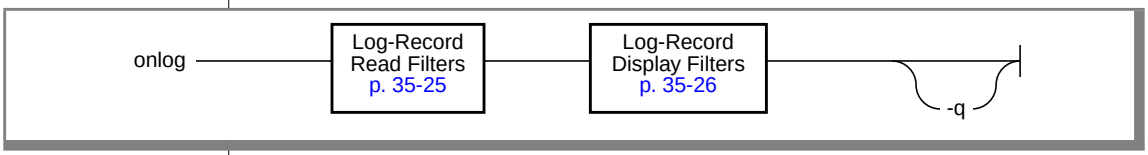
The **onlog** output is useful in debugging situations, when you want to track a specific transaction or see what changes have been made to a specific tblspace. (For information about interpreting the logical-log file contents, see Chapter 37, "Interpreting Logical-Log Records.")

Any user can run all of the **onlog** options except the **-l** option. Only user **informix** can run the **-l** option.

If the database server is in off-line mode when you execute **onlog**, only the files on disk are read. If the database server is in quiescent or on-line mode, **onlog** also reads the logical-log records stored in the logical-log buffers in shared memory (after all records on disk have been read).

When the database server reads a log file with status U (see the **-l** option of the **onstat** utility described on page 35-80) from disk while in on-line mode, the database server denies all access to the logical-log files, effectively stopping database activity for all sessions. For this reason, Informix recommends that you wait to read the contents of the logical-log files until after the files have been backed up, and then read the files from tape.

Syntax



Element	Purpose	Key Considerations
-q	Suppresses the initial header and the one-line header that appears every 18 records by default.	None.

Read Filters

You direct **onlog** to read the following portions of the logical log as it searches for records to include in the output display:

- Records stored on disk
- Records stored on tapes created by **ontape**
- Records from the specified logical-log file

By default, **onlog** displays the logical-log record header, which describes the transaction number and the record type. The record type identifies the type of operation performed.

In addition to the header, you can use the read filters to direct **onlog** to display the following information:

- Copies of blobpages from blobspaces (copied from the logical-log backup tape only, not available from disk)
- Logical-log record header and data (including copies of blobs stored in a dbspace)

Display Filters

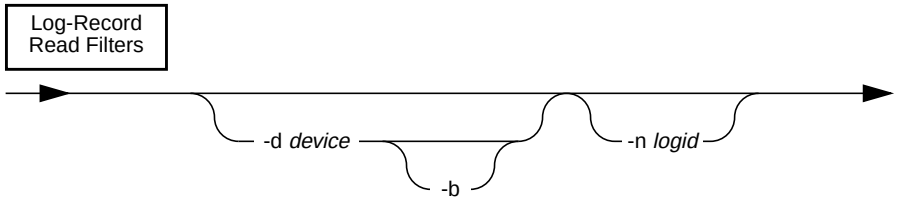
You can display every logical-log record header, or you can specify output based on the following criteria:

- Records associated with a specific table
- Records initiated by a specific user
- Records associated with a specific transaction

If an Error Is Detected

If **onlog** detects an error in the log file, such as an unrecognizable log type, it displays the entire log page in hexadecimal format and terminates.

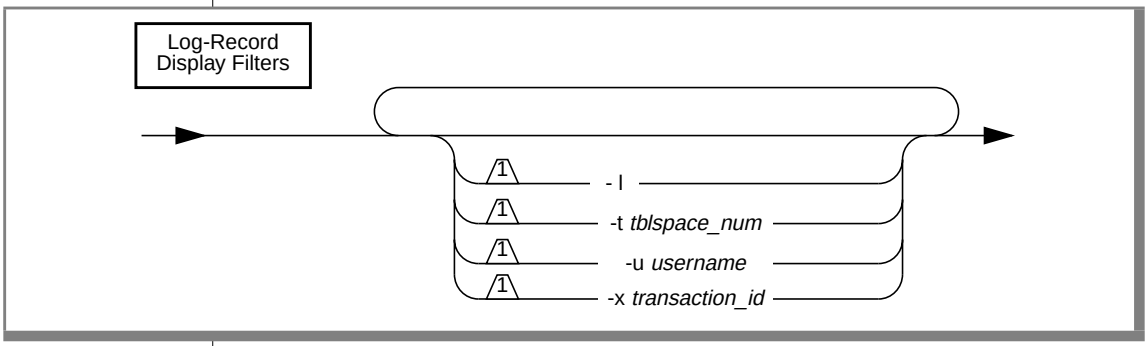
Log-Record Read Filters



Element	Purpose	Key Considerations
-b	Displays logical-log records associated with blobspace blobpages.	**Additional Information**: Dynamic Server stores these records on the logical-log backup tape as part of blobspace logging.
-d *device*	Names the pathname of the tape device where the logical-log backup tape whose contents you want to display is mounted.	**Additional Information**: The device that you name must be the same as the pathname of the device assigned to the configuration parameter LTAPEDEV. If the **-d** option is not used, **onlog** reads the logical-log files stored on disk, starting with the logical-log file with the lowest logid. **References**: For pathname syntax, see your operating-system documentation.
-n *logid*	Directs **onlog** to read only the logical-log records contained in the log file that you specify using *logid*.	**Restriction**: Value must be an unsigned integer between 3 and the value assigned to the configuration parameter LOGMAX. **Additional Information**: If you do not use the **-n** option, **onlog** reads all logical-log files available (either on disk or on tape created by ontape). To determine the logid of a particular logical-log file, use the **onstat** utility. **References**: For information about the **onstat** utility, see page 29-16.

The **onlog** utility uses the pathnames that are stored in the root dbspace reserved pages to locate the logical-log files.

Log-Record Display Filters



Element	Purpose	Key Considerations
-l	Displays the long listing of the logical-log record.	**Additional Information**: The long listing of a log record includes a complex hexadecimal and ASCII dump of the entire log record. The listing is not intended for casual use.
-t *tblspace_num*	Displays records associated with the tblspace that you specify.	**Restrictions**: Unsigned integer. Number, greater than 0, must be contained in the **partnum** column of the **systables** system catalog table. **Additional Information**: This value can be specified as either an integer or hexadecimal value. (If you do not use a `0x` prefix, the value is interpreted as an integer.) To determine the tblspace number of a particular tblspace, query the **systables** system catalog table as described on page 38-21.
-u *username*	Displays records for a specific user.	**Restrictions**: User name must be an existing login name. User name must conform to operating-system-specific rules for login name.
-x *transaction_id*	Displays only records associated with the transaction that you specify.	**Restriction**: Value must be an unsigned integer between `0` and TRANSACTIONS `-1`, inclusive. **Additional Information**: You should only need to use the **-x** option in the unlikely case that an error is generated during a rollforward. When this situation occurs, the database server sends a message to the message log that includes the transaction ID of the offending transaction. You can use this transaction ID with the **-x** option of **onlog** to investigate the cause of the error.

If you do not specify any options, **onlog** displays a short listing of all the records in the log. You can combine options with any other options to produce more selective filters. For example, if you use both the **-u** and **-x** options, **onlog** displays only the activities initiated by the specified user during the specified transaction. If you use both the **-u** and **-t** options, **onlog** displays only the activities initiated by the specified user and associated with the specified tblspace.

onmode: Mode and Shared-Memory Changes

The **onmode** flags determine which of the following operations **onmode** performs:

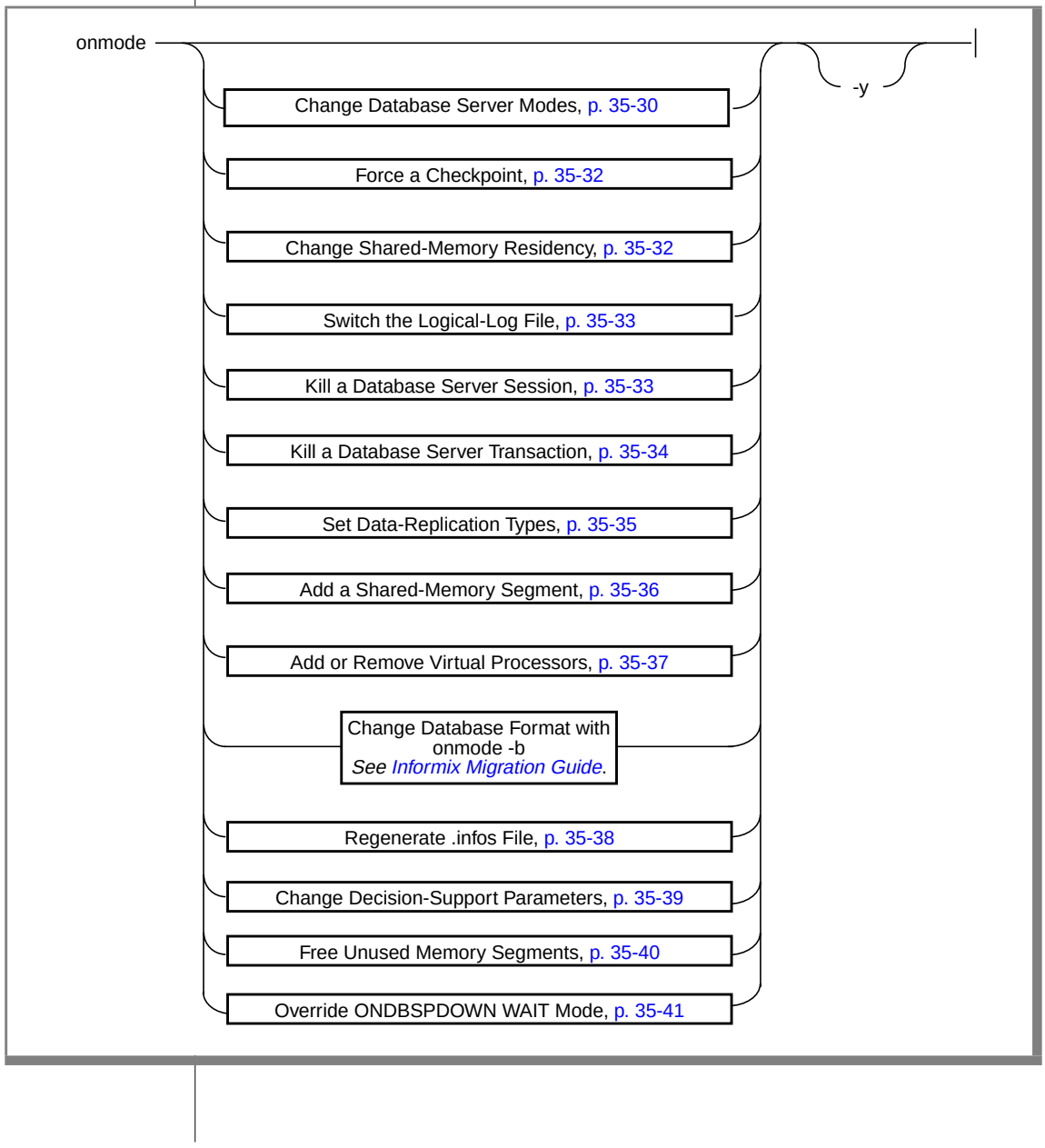
- Change the database server operating mode
- Force a checkpoint
- Change residency of the resident and virtual portions of shared memory
- Switch the logical-log file
- Kill a database server session
- Kill a database server transaction
- Set data-replication types
- Add a shared-memory segment to the virtual shared-memory portion
- Add or remove virtual processors
- Change data to the Version 7.2 database server format (to revert to that product release). For information about migrating from or reverting to earlier versions of the database server, see the *Informix Migration Guide*.
- Regenerate a **.infos** file
- Set decision-support parameters
- Free unused memory segments
- Override the WAIT mode of the ONDBSPDOWN configuration parameter

You must be user **root** or user **informix** to execute **onmode**.

UNIX

This chapter refers to ON-Monitor in many places. ON-Monitor is available only on UNIX. ♦

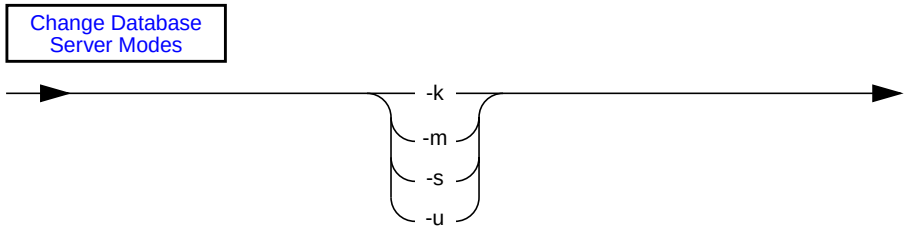
Syntax


onmode
-y
Change Database Server Modes, p. 35-30
Force a Checkpoint, p. 35-32
Change Shared-Memory Residency, p. 35-32
Switch the Logical-Log File, p. 35-33
Kill a Database Server Session, p. 35-33
Kill a Database Server Transaction, p. 35-34
Set Data-Replication Types, p. 35-35
Add a Shared-Memory Segment, p. 35-36
Add or Remove Virtual Processors, p. 35-37
Change Database Format with onmode -b
See Informix Migration Guide.
Regenerate .infos File, p. 35-38
Change Decision-Support Parameters, p. 35-39
Free Unused Memory Segments, p. 35-40
Override ONDBSPDOWN WAIT Mode, p. 35-41

Element	Purpose	Key Considerations
-y	Causes the database server to automatically respond *yes* to all prompts.	None.

If you do not use any options, the database server returns a usage statement.

Change Database Server Modes



Element	Purpose	Key Considerations
-k	Takes the database server to off-line mode and removes shared memory. See "Taking the Database Server to Off-Line Mode with the -k Option" on page 35-31.	**Additional Information**: The **-k** option is equivalent to the ON-Monitor Take-Offline option. You might want to use this option to reinitialize shared memory.b
-m	Takes the database server from quiescent to on-line mode.	**Additional Information**: The **-m** option is equivalent to the ON-Monitor On-Line option.
-s	Shuts down the database server gracefully. See "Shutting Down the Database Server Gracefully with the -s Option" on page 35-31.	**Additional Information**: The **-s** option is equivalent to the ON-Monitor Graceful-Shutdown option. Users who are using the database server are allowed to finish before the database server comes to quiescent mode, but no new connections are allowed. When all processing is finished, **-s** takes the database server to quiescent mode. The **-s** option leaves shared memory intact.
-u	Shuts down the database server immediately. See "Shutting the Database Server Down Immediately with the -u Option" on page 35-31.	**Additional Information**: The **-u** option is equivalent to the ON-Monitor Immediate-Shutdown option. This option brings the database server to quiescent mode without waiting for users to finish their sessions. Their current transactions are rolled back, and their sessions are terminated.

The options described in this section take the database server from one mode to another mode.

Taking the Database Server to Off-Line Mode with the -k Option

This option takes the database server to off-line mode and removes database-server shared memory. The **-k** option is equivalent to the ON-Monitor Take-Offline option. You might want to use this option to reinitialize shared memory.

A prompt asks for confirmation. Another prompt asks for confirmation to kill user threads before the database server comes off-line. If you want to eliminate these prompts, execute the **-y** option with the **-s** option.

Shutting Down the Database Server Gracefully with the -s Option

The **-s** option is equivalent to the ON-Monitor Graceful-Shutdown option. Users who are using the database server are allowed to finish before the database server comes to quiescent mode, but no new connections are allowed. When all processing is finished, **-s** takes the database server to quiescent mode. The **-s** option leaves shared memory intact.

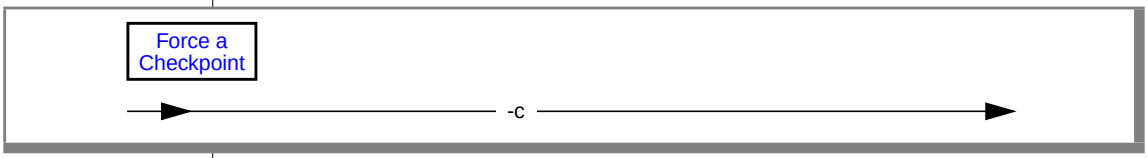
A prompt asks for confirmation. If you want to eliminate this prompt, execute the **-y** option with the **-s** option.

Shutting the Database Server Down Immediately with the -u Option

The **-u** option is equivalent to the ON-Monitor Immediate-Shutdown option. This option brings the database server to quiescent mode without waiting for users to finish their sessions. Their current transactions are rolled back, and their sessions are terminated.

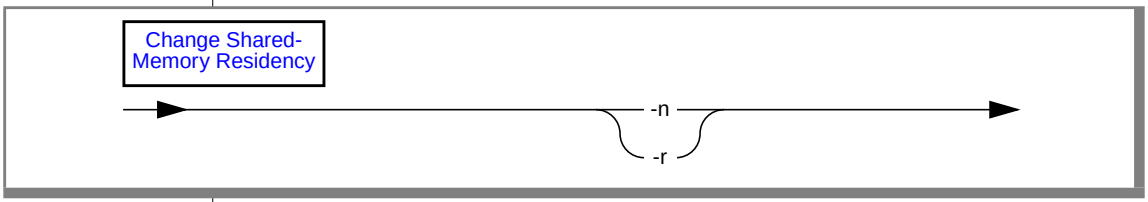
A prompt asks for confirmation. Another prompt asks for confirmation to kill user threads before the database server comes to quiescent mode. If you want to eliminate these prompts, execute the **-y** option with the **-s** option.

Force a Checkpoint



Element	Purpose	Key Considerations
-c	Forces a checkpoint.	**Additional Information**: The **-c** option is equivalent to the ON-Monitor Force-Ckpt option. You can use the **-c** option to force a checkpoint if the most-recent checkpoint record in the logical log was preventing the logical-log file from being freed (status U-B-L).

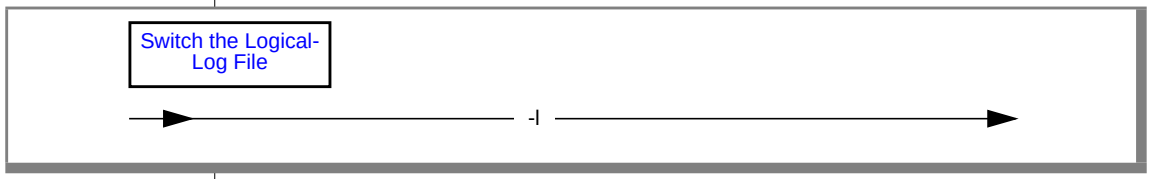
Change Shared-Memory Residency



Element	Purpose	Key Considerations
-n	Ends forced residency of the resident portion of shared memory.	**Additional Information**: This command does not affect the value of RESIDENT, the forced-residency parameter in the ONCONFIG file.
-r	Starts forced residency of the resident portion of shared memory.	**Additional Information**: This command does not affect the value of RESIDENT, the forced-memory parameter in the ONCONFIG file.

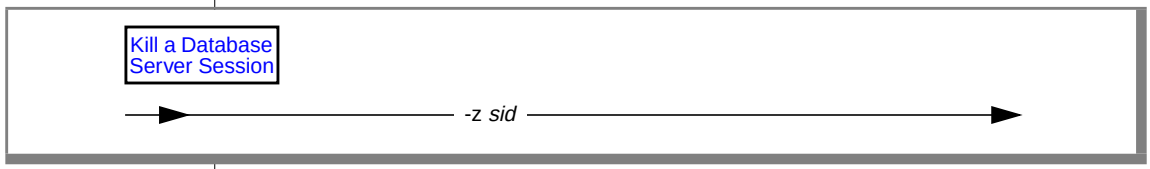
To change the forced-residency setting in the ONCONFIG configuration file, see "Turning Residency On or Off for the Next Time You Reinitialize Shared Memory" on page 12-15.

Switch the Logical-Log File



Element	Purpose	Key Considerations
-l	Switches the current logical-log file to the next logical-log file.ow	**Additional Information**: You must use **onmode** to switch to the next logical-log file. ON-Monitor has no equivalent option. **References**: For more information, see "Switching to the Next Logical-Log File" on page 19-15.

Kill a Database Server Session



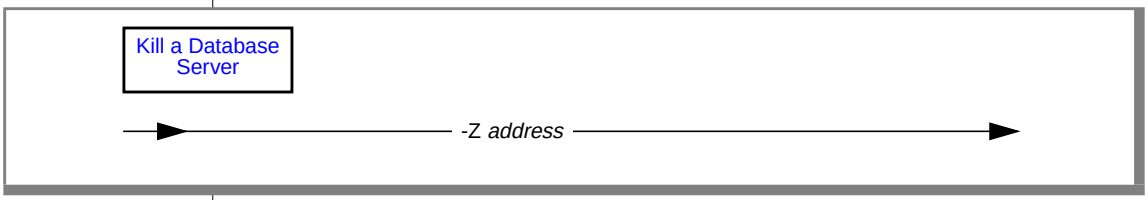
Element	Purpose	Key Considerations
-z *sid*	Kills the session that you specify in *sid*.	**Restrictions**: This value must be an unsigned integer greater than 0 and must be the session identification number of a currently running session.

To use the **-z** option, first obtain the session identification (*sessid*) with **onstat -u**, then execute **onmode -z**, substituting the session identification number for *sid*.

When you use **onmode -z**, the database server attempts to kill the specified session. If the database server is successful, it frees any resources held by the session. If the database server cannot free the resources, it does not kill the session.

If the session does not exit the section or release the latch, user **informix** or **root** can take the database server off-line, as described in "Taking the Database Server to Off-Line Mode with the -k Option" on page 35-31, to close all sessions.

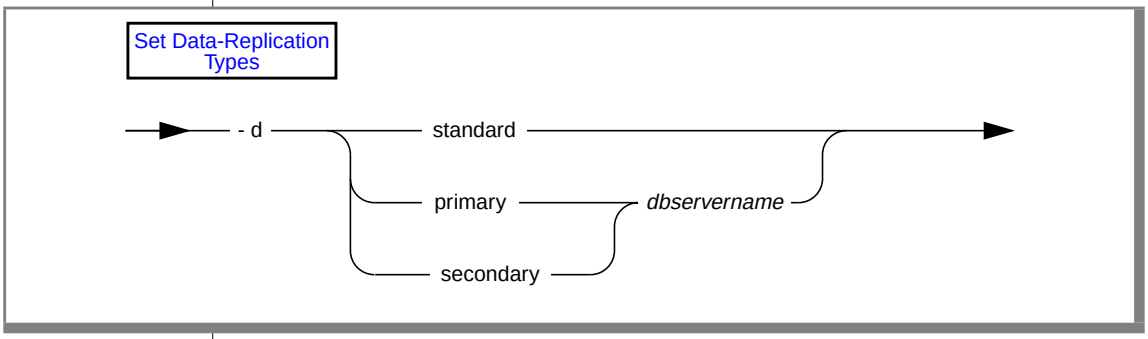
Kill a Database Server Transaction



Element	Purpose	Key Considerations
-Z *address*	Kills a distributed transaction associated with the shared-memory address *address*.	**Restrictions**: This argument must be the address of an ongoing distributed transaction that has exceeded the amount of time specified by TXTIMEOUT. The address must conform to the operating-system-specific rules for addressing shared-memory. (The address is available from **onstat -x** output.) **Additional Information**: This option is not valid until the amount of time specified by the ONCONFIG parameter TXTIMEOUT has been exceeded. The **-Z** option should rarely be used and only by an administrator of a database server involved in distributed transactions. **References**: For more information, see "Independent Actions" on page 30-18.

Warning: *If applications are performing distributed transactions, killing one of the distributed transactions can leave your client/server database system in an inconsistent state. Try to avoid this situation.*

Set Data-Replication Types



Element	Purpose	Key Considerations
-d	Used to set the data replication type, either standard, primary, or secondary, as described in the following sections.	**Restrictions**: You can use the **-d** primary and **-d** secondary options only when the database server is in quiescent mode. You can use the **-d** standard option when the database server is in quiescent, on-line, or read-only mode.
dbservername	Identifies the database server name of the primary or secondary database server.	**Restrictions**: The *dbservername* argument must correspond to the DBSERVERNAME parameter in the ONCONFIG file of the intended secondary database server. It should *not* correspond to one of the servers specified by the DBSERVERALIASES parameter. **Additional Information**: The *dbservername* argument of the other database server in the data-replication pair and the type of a database server (standard, primary, or secondary) is preserved across reinitializations of shared memory. **References**: For more information, see *range of values* for the DBSERVERNAME configuration parameter on page 33-18.

-d standard

This option drops the connection between database servers in a data replication pair (if one exists) and sets the database server type of the current database server to standard. This option does not change the mode or type of the other database server in the pair.

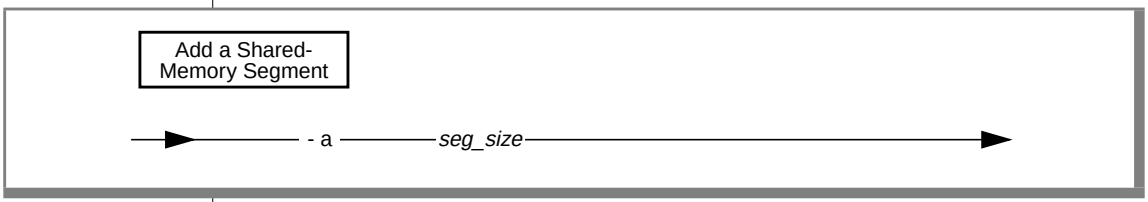
-d primary dbservername

This option sets the database server type to primary and attempts to connect with the database server specified by *dbservername*. If the connection is successful, data replication is turned on (the primary database server goes into on-line mode, and the secondary database server goes into read-only mode). If the connection is not successful, the database server comes to on-line mode, but data replication is not turned on.

-d secondary dbservername

This option sets the database server type to secondary and attempts to connect with the database server specified by *dbservername*. If the connection is successful, data replication is turned on (the primary database server goes into on-line mode, and the secondary database server goes into read-only mode). If the connection is not successful, the database server comes to read-only mode, but data replication is not turned on.

Add a Shared-Memory Segment

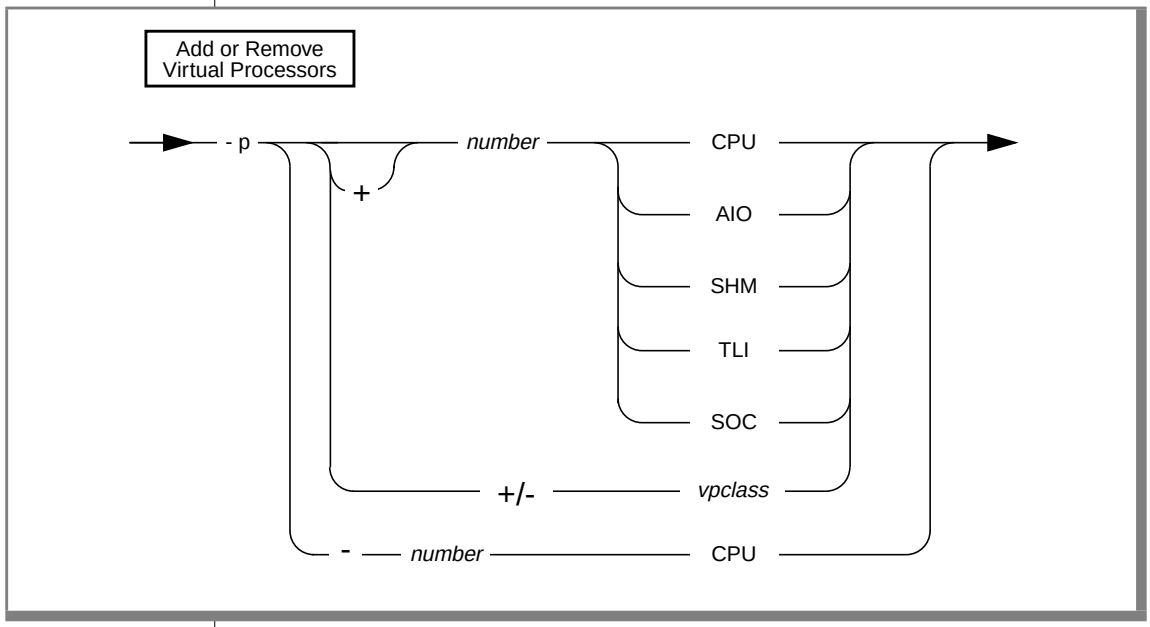


Element	Purpose	Key Considerations
-a *seg_size*	Allows you to add a new virtual shared-memory segment. Size is specified in kilobytes.	**Restrictions**: The value of *seg_size* must be a positive integer. It must not exceed the operating-system limit on the number of shared-memory segments.

Ordinarily, you do not need to add segments to the virtual portion of shared memory because the database server automatically adds segments as they are needed. However, as segments are added, the database server might reach the operating-system limit for the maximum number of segments before it acquires the memory it needs. This situation typically occurs when SHMADD is set so small that the database server exhausts the number of available segments before it acquires the memory that it needs for some operation.

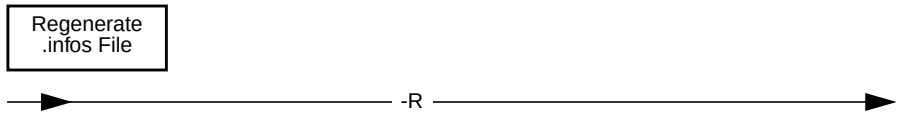
If you manually add a segment that is larger than the segment specified by SHMADD, you can avoid exhausting the operating-system limit for segments but still meet the need that the database server has for additional memory.

Add or Remove Virtual Processors



Element	Purpose	Key Considerations
-p *number*	Add or remove virtual processors. The number of virtual processors to add or remove is specified by *number*.	**Restrictions**: This value must be an unsigned integer greater than 0. You can use the **-p** option only when the database server is in on-line mode, and you can add to only one class of virtual processors at a time. **Additional Information**: If you are removing virtual processors, maximum cannot be greater than the actual number of processors of the specified type. If you are adding virtual processors, the maximum is less than 64 kilobytes, although Informix recommends that the number of CPU VPs not be greater than the number of physical processors.

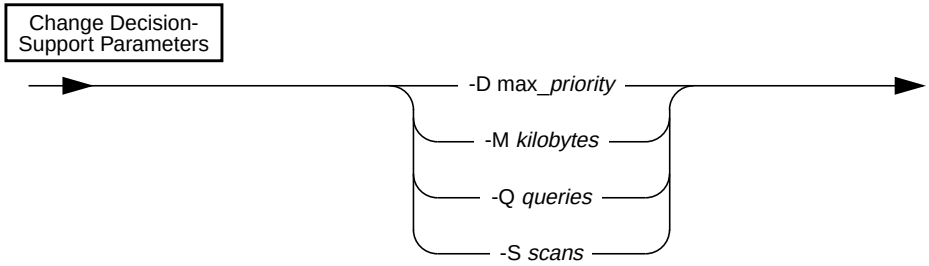
Regenerate .infos File



Element	Purpose	Key Considerations
-R	Re-creates the **.infos.dbservername** file.	**Restrictions**: Before you use the **-R** option, set the **INFORMIXSERVER** environment variable to match the DBSERVERNAME configuration parameter. Do not use the **-R** option if **INFORMIXSERVER** is one of the DBSERVERALIAS names.

When the database server accesses utilities, it uses information from the **.infos.*dbservername*** file in the **etc** directory. The database server creates and manages this file, and you should never need to do anything to the file. However, if **.infos.*dbservername*** is accidentally deleted, you must either re-create the file or reinitialize shared memory.

Change Decision-Support Parameters



Element	Purpose	Key Considerations
-D *max_priority*	Changes the value of MAX_PDQPRIORITY.	**Restrictions**: This value must be an unsigned integer between 0 and 100. **Additional Information**: Specify *max_priority* as a factor to temper the user's request for PDQ resources. **References**: For more information, see "Parameters Used for Controlling PDQ" on page 15-16 and "MAX_PDQPRIORITY" on page 33-52.
-M *kilobytes*	Changes the value of DS_TOTAL_MEMORY.	**Restrictions**: This value must be an unsigned integer between 128 * DS_MAX_QUERIES and 1,048,576. **Additional Information**: Specify *kilobytes* for the maximum amount of memory available for parallel queries. **References**: For more information, see "DS_TOTAL_MEMORY" on page 33-28 and your *Performance Guide*.

(1 of 2)

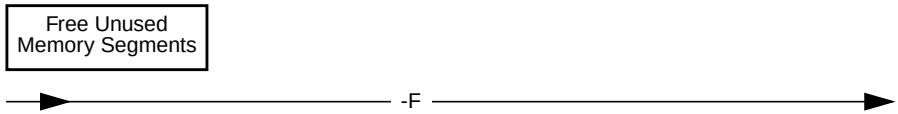
Element	Purpose	Key Considerations
-Q *queries*	Changes the value of DS_MAX_QUERIES.	**Restrictions**: This value must be an unsigned integer between 1 and 8,388,608. **Additional Information**: Specify *queries* for the maximum number of concurrently executing parallel queries. **References**: For more information, see "Parameters Used for Controlling PDQ" on page 15-16 and "DS_MAX_QUERIES" on page 33-26.
-S *scans*	Changes the value of DS_MAX_SCANS. w	**Restrictions**: This value must be an unsigned integer between 10 and 1,048,576. **Additional Information**: Specify *scans* for the maximum number of concurrently executing parallel scans. **References**: For more information, see "Parameters Used for Controlling PDQ" on page 15-16 and "DS_MAX_SCANS" on page 33-27.

(2 of 2)

These options let you change configuration parameters while the database server is on-line. The new values affect only the current instance of the database server; the values are not recorded in the ONCONFIG file. If you reinitialize shared memory, the values of the parameters revert to the values in the ONCONFIG file.

To check the values stored in the ONCONFIG file for these parameters, use **onstat -g mgm**. For more information about these configuration parameters, see Chapter 33, "Configuration Parameters."

Free Unused Memory Segments



Element	Purpose	Key Considerations
-F	Frees unused memory segments.	None.

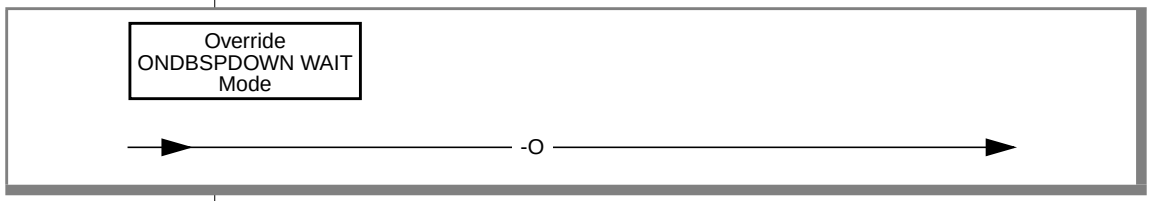
When you execute **onmode -F**, the memory manager examines each memory pool for unused memory. When the memory manager locates 8 kilobytes of contiguous unused memory, it is immediately freed. After the memory manager checks each memory pool, it begins checking memory segments and frees any that the database server no longer needs.

Informix recommends that you run **onmode -F** on a regular basis and after you direct the database server to perform any function that creates additional memory segments such as large index builds, sorts, or backups.

Running **onmode -F** causes a significant degradation of performance for any users that are active when you execute the utility. Although the execution time is brief (1 to 2 seconds), degradation for a single-user database server can reach 100 percent. Systems with multiple CPU virtual processors will experience proportionately less degradation.

To confirm that **onmode** freed unused memory, check your message log. If the memory manager frees one or more segments, it displays a message that indicates how many segments and bytes of memory were freed.

Override ONDBSPDOWN WAIT Mode



Element	Purpose	Key Considerations
-O	Overrides the WAIT mode of the ONDBSPDOWN configuration parameter.	None.

Use the **onmode -O** option only in the following circumstances:

- ONDBSPDOWN is set to WAIT.
- A disabling I/O error occurs that causes the database server to block all updating threads.
- You cannot or do not want to correct the problem that caused the disabling I/O error.
- You want the database server to mark the disabled dbspace as down and continue processing.

When you execute this option, the database server marks the dbspace responsible for the disabling I/O error as down, completes a checkpoint, and releases blocked threads. Then **onmode** prompts you with the following message:

```
This will render any dbspaces which have incurred disabling
I/O errors unusable and require them to be restored from an
archive.
Do you wish to continue?(y/n)
```

If **onmode** does not encounter any disabling I/O errors on noncritical dbspaces when you run the **-O** option, it notifies you with the following message:

```
There have been no disabling I/O errors on any non-critical
dbspaces.
```

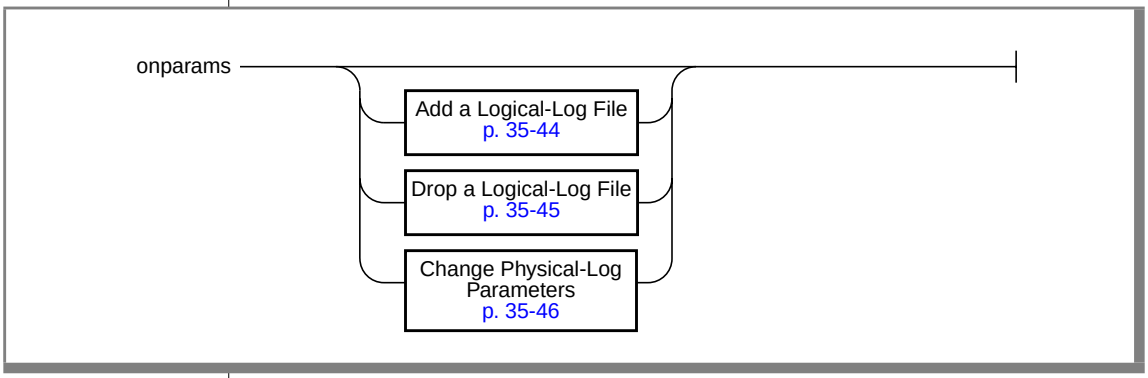
onparams: Modify Log-Configuration Parameters

The **onparams** flags determine which of the following operations **onparams** performs:

- Add a logical-log file
- Drop a logical-log file
- Change the size or location of the physical log

The database server must be in quiescent mode, and you must be logged in as user **root** or user **informix** to execute **onparams**.

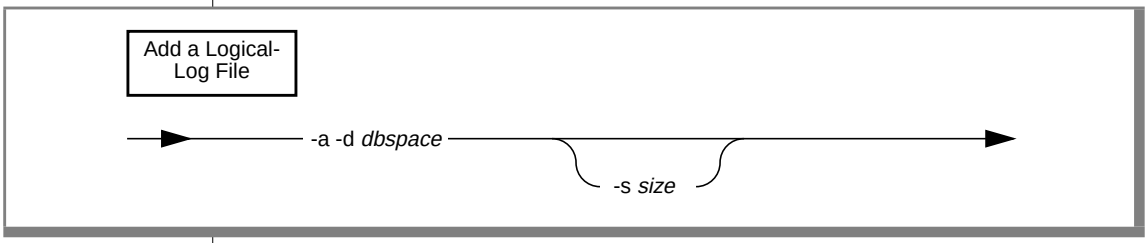
Syntax



An **onparams** command fails if a dbspace backup is in progress.

If you do not use any options, **onparams** returns a usage statement.

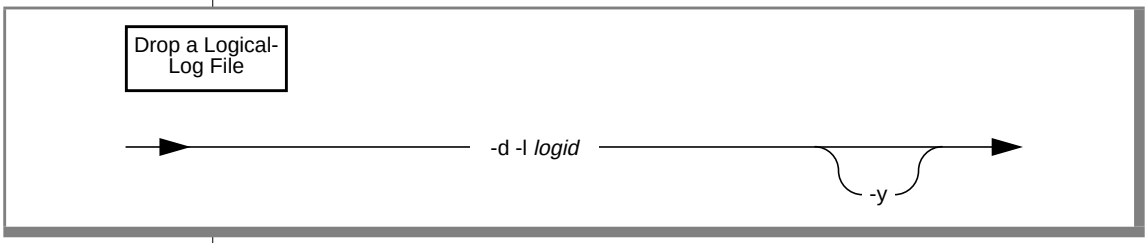
Add a Logical-Log File



Element	Purpose	Key Considerations
-a -d *dbspace*	Adds a logical-log file in the location specified by *dbspace*.	**Additional Information**: The space allocated for a logical-log file must be contiguous. The database server does not allow you to add a log file to a dbspace without adequate contiguous space. You cannot add a log file during an archive (quiescent or on-line). The newly added log file or files retain a status of A and do not become available until you create a level-0 backup. If you are using ON-Bar or ON-Archive, you only need to create a level-0 backup of the root dbspace for the log file to become available. **References:** Syntax must conform to the Identifier segment; see *Informix Guide to SQL: Syntax*.
-s *size*	Specifies a size in kilobytes for the new logical-log file.	**Restrictions**: This value must be an unsigned integer greater than or equal to 200 kilobytes. **Additional Information**: If you do not specify a size with the **-s** option, the size of the log file is taken from the value of the LOGSIZE parameter in the ONCONFIG file when database server disk space was initialized. **References**: For more information, see "Changing LOGSIZE or LOGFILES" on page 19-10.

Adding a logical-log file is one of the steps in the procedure for moving logical-log files to another dbspace. See "Moving a Logical-Log File to Another Dbspace" on page 19-8.

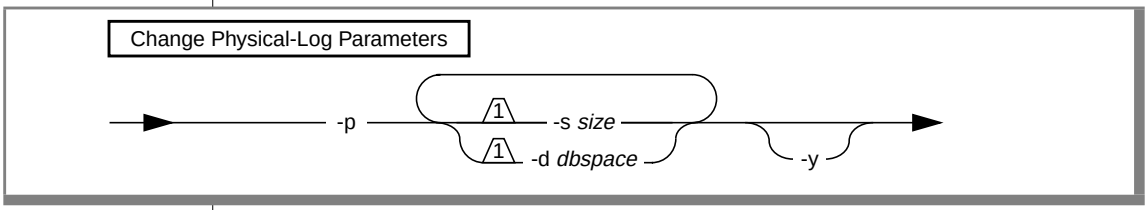
Drop a Logical-Log File



Element	Purpose	Key Considerations
-d -l *logid*	Allows you to drop a logical-log file specified by *logid*.	**Restrictions**: This value must be an unsigned integer greater than or equal to 0. **Additional Information**: You can obtain the *logid* from the number field of **onstat -l**. The database server requires a minimum of three logical-log files at all times. You cannot drop a log file if the database server is configured for three logical-log files. The database server must be in quiescent mode before you drop a logical log. Drop log files one at a time. You can only drop a log file that has a a status of Free (F) or newly Added (A). After your configuration reflects the desired number of log files, create a level-0 backup. If you are using ON-Bar or ON-Archive, you only need to create a level-0 backup of the root dbspace. Use **onstat -l** to view the status of your logical-log files.
-y	Causes the database server to automatically respond *yes* to all prompts.	None.

Dropping a logical-log file is one of the steps in the procedure for moving logical-log files to another dbspace. See "Moving a Logical-Log File to Another Dbspace" on page 19-8.

Change Physical-Log Size or Location



Element	Purpose	Key Considerations
-d *dbspace*	Changes the location of the physical log to the specified *dbspace.*	**Additional Information**: The space allocated for the physical log must be contiguous. **References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*.
-p	Changes the location or size of the physical log.	None.
-s *size*	Changes the size (in kilobytes) of the physical log.	**Restrictions**: This value must be an unsigned integer greater than or equal to 200 kilobytes. **Additional Information**: If you move the log to a dbspace without adequate contiguous space or increase the log size beyond the available contiguous space, a fatal shared-memory error occurs when you attempt to reinitialize shared memory with the new value.
-y	Causes the database server to automatically respond *yes* to all prompts.	None.

After You Change the Physical-Log Size or Location

Changes to the physical log do not take effect until you reinitialize shared memory. To reinitialize shared memory immediately, execute the command with the **-y** option.

Create a level-0 dbspace backup immediately after you reinitialize shared memory. This dbspace backup is critical for proper recovery of the database server. If you are using ON-Bar or ON-Archive, you only need to create a level-0 dbspace backup of the root dbspace.

If you move the log to a dbspace without adequate contiguous space, a fatal shared-memory error occurs when you attempt to reinitialize shared memory with the new value.

For more information, see "Changing the Physical-Log Location and Size" on page 21-3.

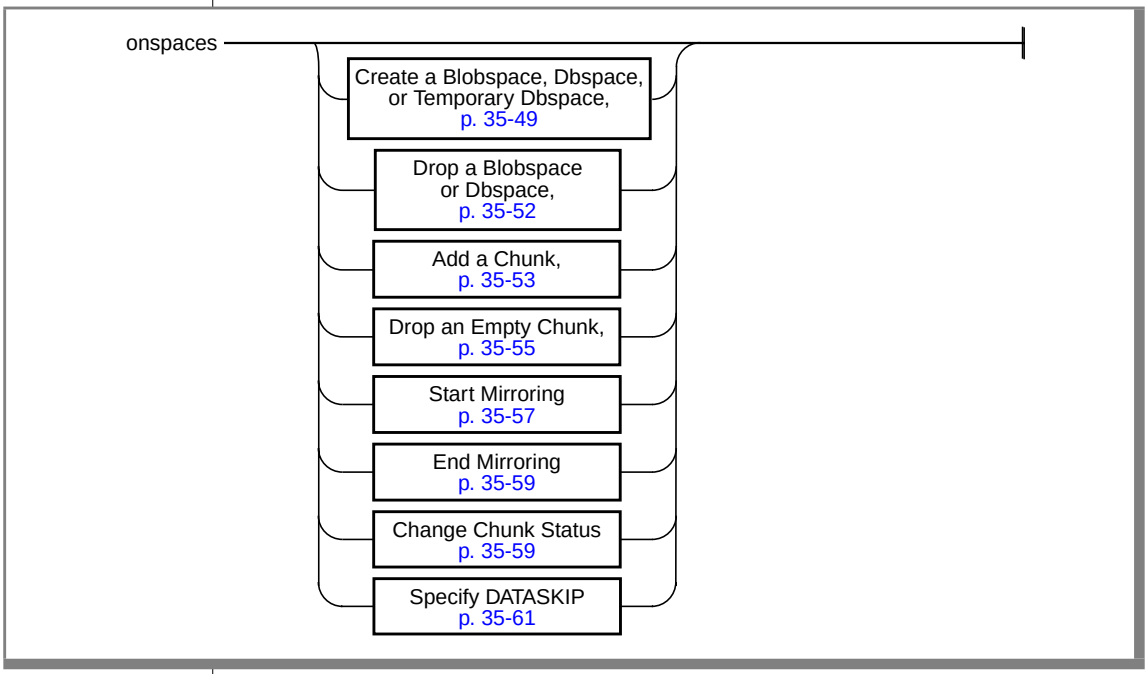
onspaces: Manage Database Spaces

The **onspaces** utility lets you perform the following tasks:

- Create a blobspace, dbspace, or temporary dbspace
- Drop a blobspace or dbspace
- Add a chunk
- Drop a chunk
- Start mirroring
- End mirroring
- Change chunk status
- Set the DATASKIP parameter

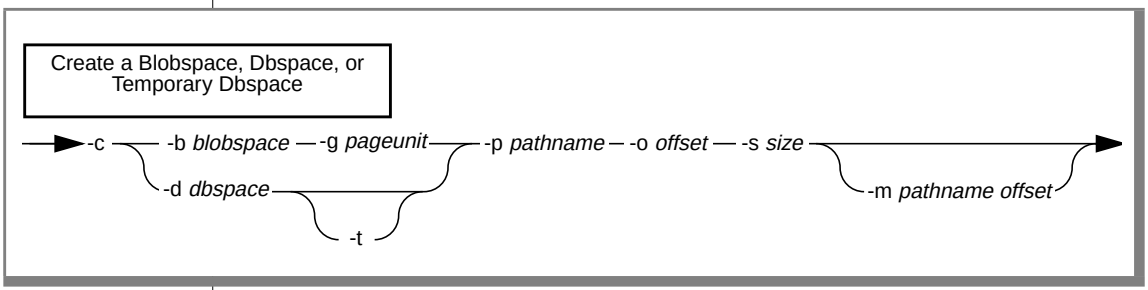
You must be logged in as user **root** or user **informix** to execute **onspaces**.

Syntax



Create a Blobspace, Dbspace, or Temporary Dbspace

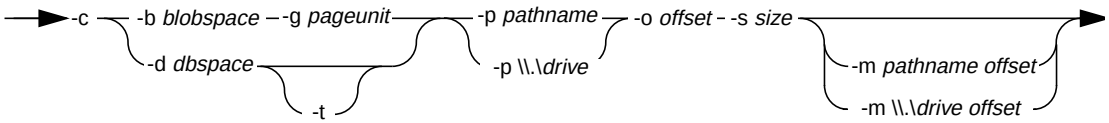
UNIX



-

WIN NT

Create a Blobspace, Dbspace, or Temporary Dbspace



◆

Element	Purpose	Key Considerations
-b *blobspace*	Names the blobspace to be created.	**Restrictions**: See the restrictions in "Creating a Blobspace" on page 14-16. **References**: The syntax must conform to the Identifier segment. For more information, see the *Informix Guide to SQL: Syntax*.
-c	Creates a blobspace or dbspace.	**References**: For more information, see "Creating a Dbspace" on page 14-11, or "Creating a Blobspace" on page 14-16.
drive (Windows NT only)	Specifies the Windows NT drive to allocate as unbuffered disk space. The format can be either \\.\<*drive*>, where *drive* is the drive letter assigned to a disk partition, or \\.*PhysicalDrive*<*number*>, where *PhysicalDrive* is a constant value and *number* is the physical drive number.	**References:** For information on allocating unbuffered disk space on Windows NT platforms, see "Allocating Raw Disk Space on Windows NT" on page 14-9.
-d *dbspace*	Names the dbspace to be created.	**Restrictions**: See the restrictions in "Creating a Dbspace" on page 14-11. **References**: The syntax must conform to the Identifier segment. For more information, see the *Informix Guide to SQL: Syntax*.

(1 of 2)

Element	Purpose	Key Considerations
-g *pageunit*	Specifies the blobspace blobpage size in terms of *page_unit*, the number of disk pages per blobpage.	**Restrictions**: Unsigned integer. Value must be greater than 0. **References**: For more information, see "Determining Database Server Page Size" on page 14-17.
-m *pathname offset*	Specifies an optional pathname and offset to the chunk that will mirror the initial chunk of the new blobspace or dbspace. Also see the entries for **-p** *pathname* and **-o** *offset* in this table.	**References**: For more information, see "Creating a Dbspace" on page 14-11 and "Creating a Blobspace" on page 14-16.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the new blobspace or dbspace.	**Restrictions**: Unsigned integer. Value must be greater than 0. **References**: For more information, see "Allocating Raw Disk Space on UNIX" on page 14-8.
-p *pathname*	Indicates the disk partition or device of the initial chunk of the blobspace or dbspace that you are creating.	**Additional Information**: The chunk must be an existing unbuffered device or buffered file. When you specify a pathname, you can use either a full pathname or a relative pathname. However, if you use a relative pathname, it must be relative to the directory that was the current directory when you initialized the database server. **References**: For pathname syntax, see your operating-system documentation.
-s *size*	Indicates, in kilobytes, the size of the initial chunk of the new blobspace or dbspace.	**Restrictions**: Unsigned integer. Value must be greater than 0. Size must not exceed 2 gigabytes.
-t	Creates a temporary dbspace.	**References**: For more information, see "Creating a Temporary Dbspace with the -t Option" on page 35-52 and "What Is a Temporary Dbspace?" on page 13-20.

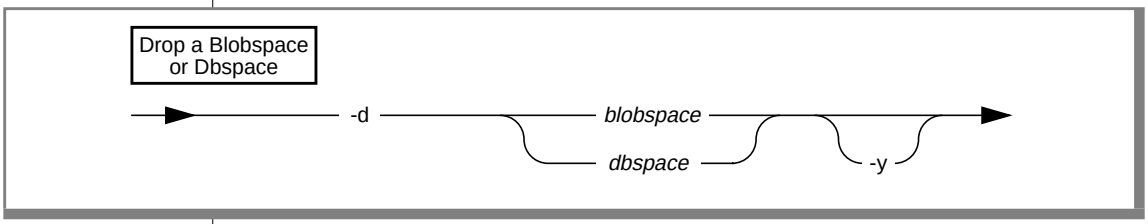
(2 of 2)

Creating a Temporary Dbspace with the -t Option

When you create a temporary dbspace with ON-Monitor (UNIX only) or **onspaces**, the database server does not use the newly created temporary dbspace until you perform the following steps:

- Add the name of the new temporary dbspace to your list of temporary dbspaces in the DBSPACETEMP configuration parameter, the **DBSPACETEMP** environment variable, or both.
- Reinitialize the database server.

Drop a Blobspace or Dbspace

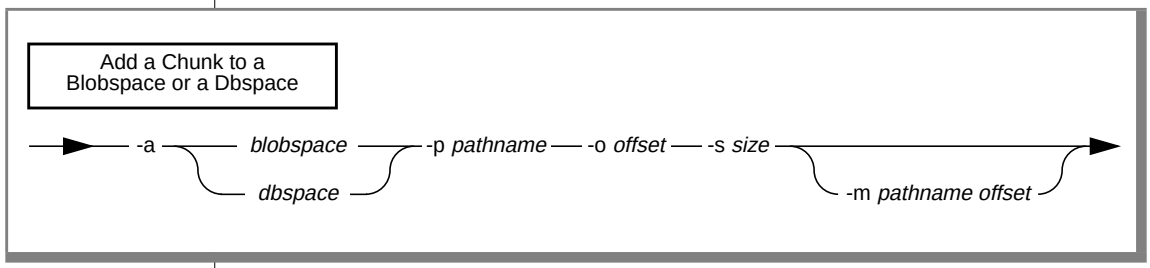


Element	Purpose	Key Considerations
-d	Indicates that a blobspace or dbspace is to be dropped.	**Restriction**: Execute **oncheck -pe** to verify that no table is currently storing data in the blobspace or dbspace. **References**: For more information, see "Dropping a Dbspace or Blobspace" on page 14-19.
-y	Causes the database server to automatically respond *yes* to all prompts.	None.
blobspace	Names the blobspace to be dropped.	**Additional Information**: Before you drop a blobspace, drop all tables that include a TEXT or BYTE column that references the blobspace.
dbspace	Names the dbspace to be dropped.	**Additional Information**: Before you drop a dbspace, drop all databases and tables that you previously created in the dbspace.

Important: *Do not specify a pathname when you are dropping a dbspace or blobspace.*

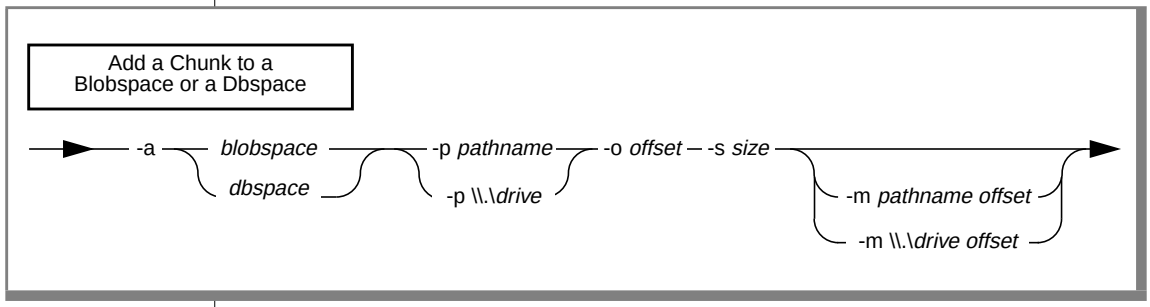
Add a Chunk to a Blobspace or Dbspace

UNIX



♦

WIN NT



♦

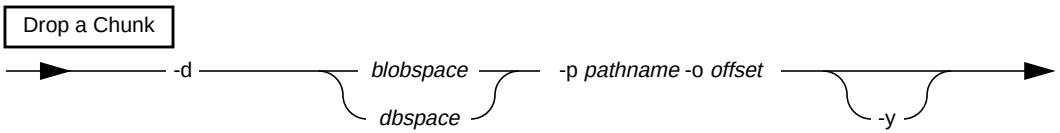
Element	Purpose	Key Considerations
-a	Indicates that a chunk is to be added.	None.
drive (Windows NT only)	Specifies the Windows NT drive to allocate as unbuffered disk space. The format can be either \\.\<*drive*>, where *drive* is the drive letter assigned to a disk partition, or \\.*PhysicalDrive*<*number*>, where *PhysicalDrive* is a constant value and *number* is the physical drive number.	**References:** For more information on allocating unbuffered disk space on Windows NT platforms, see "Allocating Raw Disk Space on Windows NT" on page 14-9.
-m *pathname offset*	Specifies an optional pathname and offset to the chunk that will mirror the new chunk. Also see the entries for *pathname* and *offset* in this table.	**References**: For more information, see "Adding a Chunk to a Dbspace" on page 14-14 and "Adding a Chunk to a Blobspace" on page 14-18.
-o *offset*	After the **-a** option, *offset* indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the new blobspace or dbspace.	**Restrictions**: Unsigned integer. Value must be greater than 0. **References**: Syntax must conform to the Identifier segment; see *Informix Guide to SQL: Syntax*. For more information, see "Allocating Raw Disk Space on UNIX" on page 14-8.
-p *pathname*	Indicates the disk partition or device of the initial chunk of the blobspace or dbspace that you are adding.	**Additional Information**: The chunk must be an existing unbuffered device or buffered file. When you specify a pathname, you can use either a full pathname or a relative pathname. However, if you use a relative pathname, it must be relative to the directory that was the current directory when you initialized the database server. **References**: For pathname syntax, see your operating-system documentation.

(1 of 2)

Element	Purpose	Key Considerations
-s *size*	Indicates, in kilobytes, the size of the initial chunk of the new blobspace or dbspace.	**Restrictions**: Unsigned integer. Value must be greater than 0. Size must not exceed 2 gigabytes.
blobspace	Names the blobspace to which you are adding a chunk.	**Restrictions**: See "Adding a Chunk to a Blobspace" on page 14-18. **References:** Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*.
dbspace	Names the dbspace to which you are adding a chunk.	**Restrictions**: See "Adding a Chunk to a Dbspace" on page 14-14. **References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*.

(2 of 2)

Drop a Chunk



Element	Purpose	Key Considerations
-d	Drops a chunk.	**Restrictions**: See "Dropping a Chunk from a Dbspace with onspaces" on page 14-18 or "Dropping a Chunk from a Blobspace" on page 14-19.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the blobspace or dbspace that you are dropping.	**Restrictions**: Unsigned integer. Value must be greater than or equal to 0. **References**: For more information, see "Allocating Raw Disk Space on UNIX" on page 14-8.

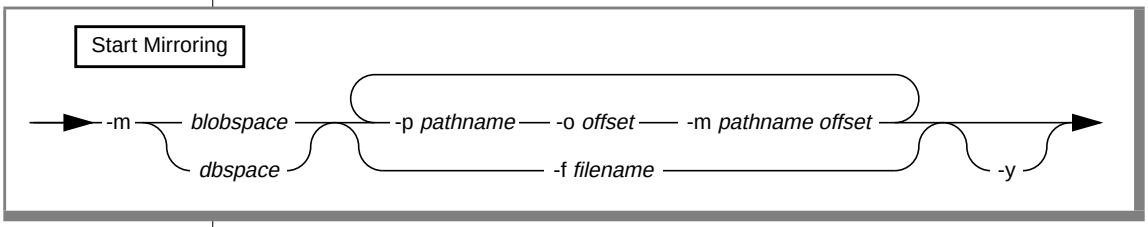
(1 of 2)

Element	Purpose	Key Considerations
-p *pathname*	Indicates the disk partition or device of the initial chunk of the blobspace or dbspace that you are dropping.	**Additional Information**: The chunk must be an existing unbuffered device or buffered file. When you specify a pathname, you can use either a full pathname or a relative pathname. However, if you use a relative pathname, it must be relative to the directory that was the current directory when you initialized the database server. **References**: For pathname syntax, see your operating-system documentation.
-y	Causes the database server to automatically respond *yes* to all prompts.	None.
blobspace	Names the blobspace from which the chunk will be dropped.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "Dropping a Chunk from a Blobspace" on page 14-19.
dbspace	Names the dbspace from which the chunk will be dropped.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "Dropping a Chunk from a Dbspace with onspaces" on page 14-18.

(2 of 2)

Important: *You must specify a pathname to indicate to the database server that you are dropping a chunk.*

Start Mirroring



Element	Purpose	Key Considerations
-f *filename*	Indicates that chunk-location information is in a file named *filename*.	**Additional Information**: The file must be a buffered file that already exists. The pathname must conform to the operating-system-specific rules for pathnames. **References**: For more information, see "Using a File to Specify Chunk-Location Information with the -f Option" on page 35-58.
-m	Adds mirroring for an existing dbspace or blobspace.	None.
-m *pathname offset*	The second time *pathname* occurs in the syntax diagram, it indicates the disk partition or device of the initial chunk of the dbspace or blobspace that performs the mirroring. The second time *offset* appears in the syntax diagram, it indicates the offset to reach the mirrored chunk of the newly mirrored dbspace or blobspace. Also see the entries for *pathname* and *offset* in this table.	None.
-o *offset*	The first time *offset* occurs in the syntax diagram, it indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the newly mirrored dbspace or blobspace.	**Restrictions**: Unsigned integer. Value must be greater than 0. **References**: For more information, see "Allocating Raw Disk Space on UNIX" on page 14-8.

(1 of 2)

Element	Purpose	Key Considerations
-p *pathname*	The first time *pathname* occurs in the syntax diagram, it indicates the disk partition or device of the initial chunk of the dbspace or blobspace that you want to mirror.	**Additional Information**: The chunk must be an existing unbuffered device or buffered file. When you specify a pathname, you can use either a full pathname or a relative pathname. However, if you use a relative pathname, it must be relative to the directory that was the current directory when you initialized the database server. **References**: For pathname syntax, see your operating-system documentation.
-y	Causes the database server to automatically respond *yes* to all prompts.	None.
blobspace	Names the blobspace that you want to mirror.	**References**: Syntax must conform to the Identifier segment; see *Informix Guide to SQL: Syntax*. For more information, see "Starting Mirroring" on page 24-6.
dbspace	Names the dbspace that you want to mirror.	**References**: Syntax must conform to the Identifier segment; see *Informix Guide to SQL: Syntax*. For background information, see "Starting Mirroring" on page 24-6.

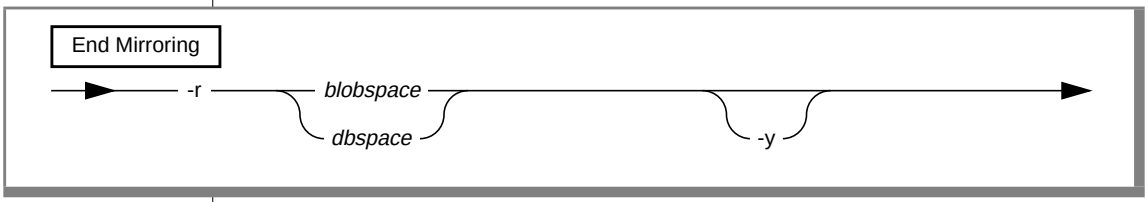
(2 of 2)

Using a File to Specify Chunk-Location Information with the -f Option

You can create a file that contains the chunk-location information. Then, when you execute **onspaces**, use the **-f** option to indicate to the database server that this information is contained in a file whose name you specify in *filename*.

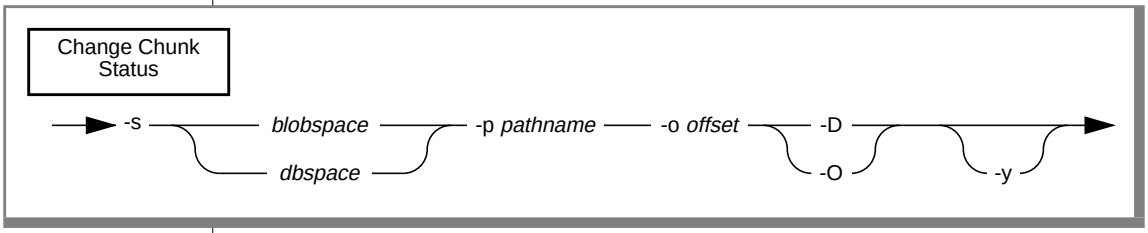
If the dbspace that you are mirroring contains multiple chunks, you must specify a mirrored chunk for each of the primary chunks in the dbspace that you want to mirror. For an example that enables mirroring for a multichunk dbspace, see "Starting Mirroring for Unmirrored Dbspaces with onspaces" on page 24-8.

End Mirroring



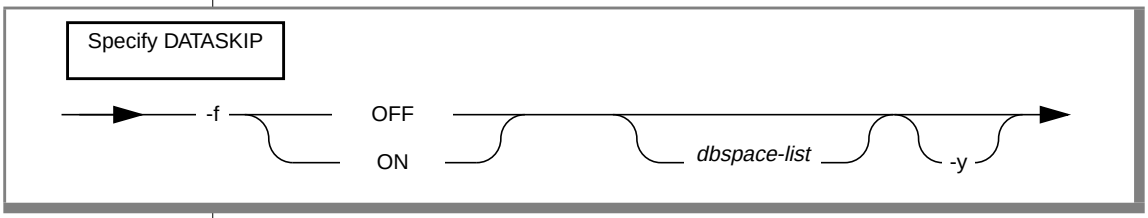
Element	Purpose	Key Considerations
-r	Indicates to the database server that mirroring should be ended for an existing dbspace or blobspace.	**References**: For more information, see "Ending Mirroring" on page 24-12.
-y	Causes the database server to respond *yes* to all prompts automatically.	None.
blobspace	Names the blobspace for which you want to end mirroring.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "Ending Mirroring" on page 24-12.
dbspace	Names the dbspace for which you want to end mirroring.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "Ending Mirroring" on page 24-12.

Change Status of a Mirrored Chunk



Element	Purpose	Key Considerations
-D	Indicates that you want to take the chunk down.	None.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the chunk.	**Restrictions**: Unsigned integer. Value must be greater than or equal to 0. **References**: For more information, see "Allocating Raw Disk Space on UNIX" on page 14-8.
-O	Indicates that you want to restore the chunk and bring it on-line.	None.
-p *pathname*	Indicates the disk partition or device of the chunk.	**Additional Information**: The chunk can be an unbuffered device or a buffered file. When you specify a pathname, you can use either a full pathname or a relative pathname. However, if you use a relative pathname, it must be relative to the directory that was the current directory when you initialized the database server. **References**: For pathname syntax, see your operating-system documentation.
-s	Indicates that you want to change the status of a chunk.	**Restrictions**: You can only change the status of a chunk in a mirrored pair. **References**: For more information, see "Changing the Mirror Status" on page 24-10.
-y	Causes the database server to respond *yes* to all prompts automatically.	None.
blobspace	Names the blobspace whose status you want to change.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "Changing the Mirror Status" on page 24-10.
dbspace	Names the dbspace whose status you want to change.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "Changing the Mirror Status" on page 24-10.

Specify DATASKIP Parameter



Element	Purpose	Key Considerations
-f	Indicates to the database server that you want to change the DATASKIP default for specified dbspaces or all dbspaces.	**Additional Information**: All changes in the system-wide DATASKIP status are recorded in the message log.
-y	Causes the database server to automatically respond *yes* to all prompts.	None.
dbspace-list	Specifies the name of one or more dbspaces for which DATASKIP will be turned ON or OFF.	**References**: Syntax must conform to the Identifier segment; see the *Informix Guide to SQL: Syntax*. For more information, see "DATASKIP" on page 33-16 and your *Performance Guide*.
OFF	Turns off DATASKIP.	**Additional Information**: If OFF is used without *dbspace-list*, DATASKIP is turned off for all fragments. If OFF is used with *dbspace-list*, only the specified fragments are set with DATASKIP off.
ON	Turns on DATASKIP.	**Additional Information**: If ON is used without *dbspace-list*, DATASKIP is turned on for all fragments. If ON is used with *dbspace-list*, only the specified fragments are set with DATASKIP on.

The **onspaces** utility lets you specify DATASKIP on a dbspace level or across all dbspaces.

onstat: Monitor Database Server Operation

The **onstat** utility reads shared-memory structures and provides statistics about the database server that are accurate at the instant that the command executes. The *system-monitoring interface* also provides information about the database server. For information on the system-monitoring interface, see Chapter 34, "The sysmaster Database."

The contents of shared memory might change as the **onstat** output displays. The **onstat** utility does not place any locks on shared memory, so running the utility does not affect performance. For information about disk usage and data storage, see "oncheck: Check, Repair, or Display" on page 35-5.

The following table lists each **onstat** option flag and its function.

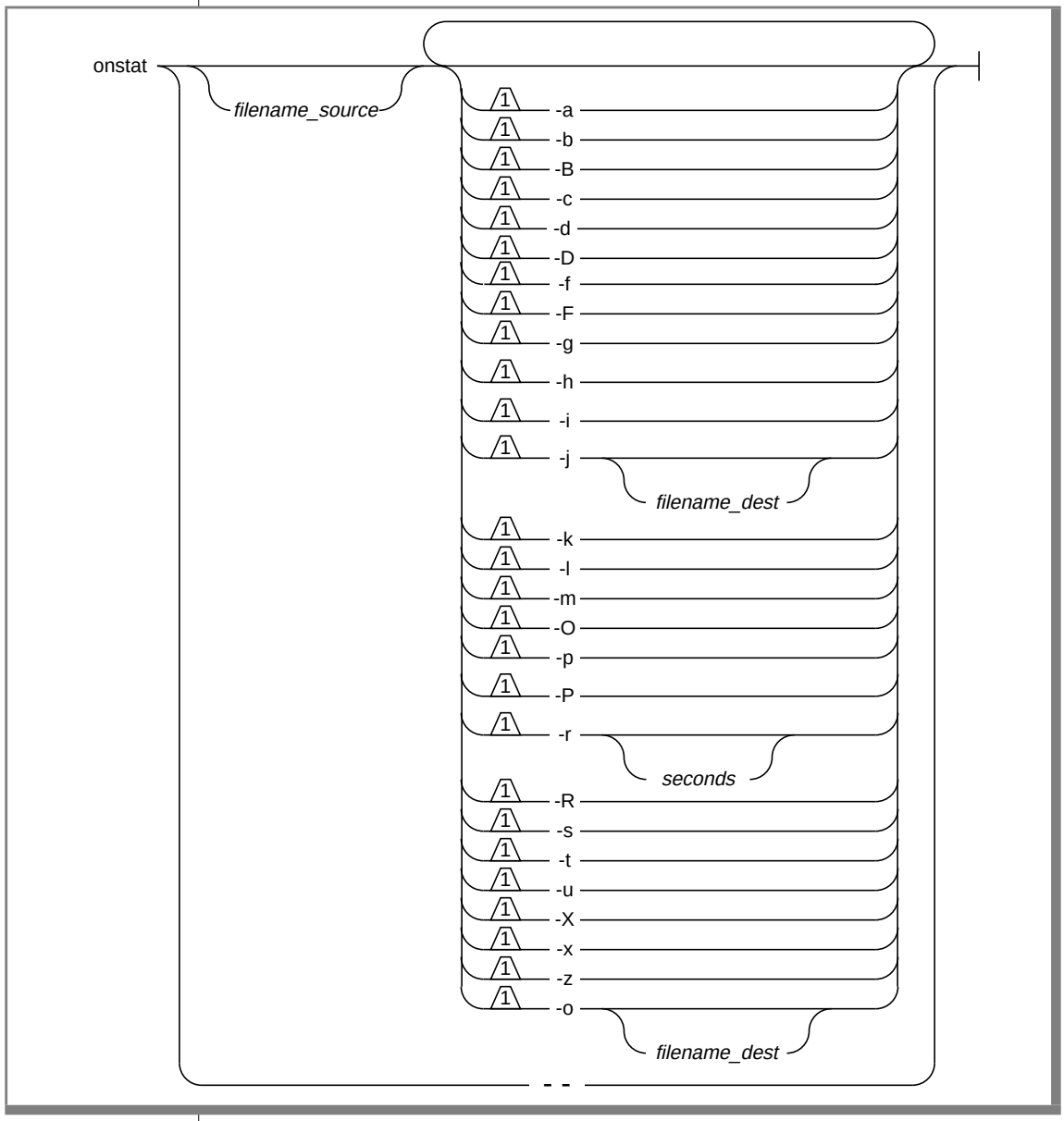
Topic or Function	Option Flag
All **onstat** options	--
Btree cleaner requests	-C
Buffers, all (in use or not)	-B
Buffers in use	-b
Buffers, includes addresses of waiting threads	-X
Buffer hash chain info	-h
Configuration-file information (ONCONFIG file)	-c
DATASKIP information	-f
Dbspace chunks, general information	-d
Dbspace chunks, page reads / writes	-D
Interactive mode	-i
Latches	-s
Locks held	-k
Logging information (logical and physical logs, including page addresses)	-l

(1 of 2)

Topic or Function	Option Flag
LRU queues	-R
Monitoring information	-g
Dynamic Server message log	-m
Dynamic Server profile of activity	-p
Optical Subsystem memory cache and staging-area blobspace information	-O
Repeat this **onstat** command periodically	-r
Shared-memory segment (save it to a file)	-o
Summary of user-oriented (lowercase) options	-a
Tblspaces, active	-t
Transaction information	-x
User threads and transactions	-u
Write type statistics (gathered when pages are flushed)	-F
Zero out all statistic counts	-z

(2 of 2)

Syntax


onstat
filename_source
1
-a
-b
-B
-c
-d
-D
-f
-F
-g
-h
-i
-j
filename_dest
-k
-l
-m
-O
-p
-P
-r
seconds
-R
-s
-t
-u
-X
-x
-z
-o
filename_dest
- -

Element	Purpose	Key Considerations
--	Displays a listing of all onstat options and their functions. See "-- Option" on page 35-70.	**Additional Information**: This option is the only option flag that you cannot combine with any other flag.
-a	Interpreted as **onstat -cuskbtdlp** (all lowercase option flags). Displays output in that order. See "-a Option" on page 35-70.	None.
-b	Displays information about buffers currently in use, including number of resident pages in the buffer pool. See "-b Option" on page 35-70.	None.
-B	Obtains information about all Dynamic Server buffers, not just buffers currently in use. See the entry for **-b** in this table.	**Additional Information**: The **-B** output display fields are the same as the fields that appear in the **-b** output.
-c	Displays the ONCONFIG file.	**Additional Information**: Dynamic Server first checks if you have assigned a value to the environment variable **ONCONFIG**. If so, Dynamic Server displays the contents of the specified by the **ONCONFIG** environment variable. If not, Dynamic Server displays, by default, the contents of **$INFORMIXDIR/etc/onconfig** on UNIX or **%INFORMIXDIR%\etc\onconfig** on Windows NT.
-C	Prints B-tree cleaner information.	None.
-d	Displays information for chunks in each space. See "-d Option" on page 35-72.	None.
-D	Displays page-read and page-write information for the first 50 chunks in each dbspace. See "-D Option" on page 35-74.	None.
-f	Lists the dbspaces currently affected by the DATASKIP feature. See "-f Option" on page 35-74.	None.
-F	Displays a count for each type of write that flushes pages to disk. See "-F Option" on page 35-75.	None.
-g	Monitoring options. See "-g Monitoring Options" on page 35-76.	None.

(1 of 3)

Element	Purpose	Key Considerations
-h	Prints buffer hash chain information.	None.
-i	Puts the **onstat** utility into interactive mode. See "-i Option" on page 35-79.	None.
-k	Displays information about active locks. See "-k Option" on page 35-79.	None.
-l	Displays information about physical and logical logs. See "-l Option" on page 35-80.	None.
-m	Displays the 20 most-recent lines of the system message log. See "-m Option" on page 35-82.	**Additional Information**: Output from this option lists the full pathname of the message-log file and the 20 file entries. A date-and-time header separates the entries for each day. A time stamp prefaces single entries within each day. The name of the message log is specified as MSGPATH in the ONCONFIG file.
-o	Saves a copy of the shared-memory segment to *filename*.	**Additional Information**: If you omit a filename in the **onstat** command, the copy of shared memory is saved to **onstat.out** in the current directory.
-O	Displays information about the Optical Subsystem memory cache and staging-area blobspace. See "-O Option" on page 35-82.	None.
-p	Displays profile counts. See "-p Option" on page 35-84.	None.
-P	Displays for all partitions the partition number and the break up of pages in the buffer pool belonging to the partition.	None.
-r	Causes the accompanying **onstat** options to execute repeatedly after they wait the specified *seconds* between each execution. The default value of *seconds* is 5.	**Additional Information**: To end execution, press DEL or CTRL-C.
-R	Displays detailed information about the LRU queues, FLRU queues, and MLRU queues. See "-R Option" on page 35-87.	None.

(2 of 3)

Element	Purpose	Key Considerations
-s	Displays general latch information. See "-s Option" on page 35-88.	None.
-t	Displays tblspace information for active tblspaces, including residency state. See "-t Option" on page 35-89.	None.
-u	Prints a profile of user activity. See "-u Option" on page 35-90.	None.
-x	Displays information about transactions. See "-x Option" on page 35-92.	None.
-X	Obtains precise information about the threads that are sharing and waiting for buffers. See "-X Option" on page 35-94.	None.
-z	Sets the profile counts to zero. See "-z Option" on page 35-94.	None.
filename_dest	Specifies destination file that will contain the copy of the shared-memory segment.	**Restrictions**: Name must not match the name of any existing file. **References**: For pathname syntax, see your operating-system documentation.
filename_source	Specifies file that **onstat** will read as source for the requested information.	**Restrictions**: This file must include a previously stored shared-memory segment that you created using the **-o** option of **onstat**. **References**: For specific details on this option, see "Statistics Culled from Source File." For pathname syntax, see your operating-system documentation.
seconds	Specifies number of seconds between each execution of the **onstat -r** command.	**Restrictions**: This value must be an unsigned integer greater than 0.

(3 of 3)

Statistics Culled from Source File

Use the *filename_source* parameter with other option flags to derive the requested **onstat** statistics from the shared-memory segment contained in *filename_source*. You must first use the **onstat -o** command to create a file that contains a shared-memory segment.

Interactive Execution

You can put the **onstat** utility in interactive mode with the **-i** option. Interactive mode allows you to enter multiple options, one after the other, without exiting the program. For information on using interactive mode, see "-i Option" on page 35-79.

Continuous onstat Execution

Use the *seconds* parameter with the **-r** option flag to cause all other flags to execute repeatedly after they wait the specified seconds between each execution.

Output Header

All **onstat** output includes a header. The header takes the following form:

```
Version--Mode (Type)--(Checkpnt)--Up Uptime--Sh_mem Kbytes
```

Version is the product version number.

Mode is the current operating mode.

(Type) is the data-replication type of the database server. If the database server is not involved in data replication, this field does not appear. If the type is primary, the value `P` appears. If the type is secondary, the value `S` appears.

(Checkpnt) is a checkpoint flag. If it is set, the header might display two other fields after the mode if the timing is appropriate:

- (CKPT REQ) indicates that some Dynamic Server user thread has requested a checkpoint.
- (CKPT INP) indicates that a checkpoint is in progress. During the checkpoint, access is limited to read only. Dynamic Server cannot write or update data until the checkpoint ends.

Uptime indicates how long the database server has been running.

Sh_mem is the size of Dynamic Server shared memory, expressed in kilobytes.

A sample header follows:

```
Informix Dynamic Server Version 7.30.UC1--On-Line--Up
15:11:41--368 KBytes
```

Logs Full Sub-Header

If the database server is blocked, the **onstat** header output includes an extra line that reads as follows:

```
Blocked: reason(s)
```

The reason can be one or more of the following.

Reason	Explanation
CKPT	Checkpoint
LONGTX	Long transaction
ARCHIVE	Ongoing dbspace backup
MEDIA_FAILURE	Media failure
HANG_SYSTEM	Dynamic Server failure
DBS_DROP	Dropping a dbspace
DDR	Discrete data replication
LBU	Logs full high-water mark

Option Descriptions

You can combine multiple **onstat** option flags in a single command.

No Options

If you invoke **onstat** without any options, the command is interpreted as **onstat -pu** (**-p** option and **-u** option).

-- Option

The **--** option displays a listing of all **onstat** options and their functions. This option is the only option flag that you cannot combine with any other flag.

-a Option

The **-a** option is interpreted as **onstat -cuskbtdlp** (all lowercase option flags), and output is displayed in that order. For an explanation of each option, refer to the appropriate flag in the paragraphs that follow.

-b Option

The **-b** option displays information about buffers currently in use, including the total number of resident pages in the buffer pool. (Refer to **onstat -B** for information about all buffers, not just those in use. Refer to **onstat -g iob** for information about the use of big buffers.) You can interpret output from the **-b** option as follows:

address — is the address of the buffer header in the buffer table.

userthread — is the address of the most-recent user thread to access the buffer table. Many user threads might be reading the same buffer concurrently.

flgs — describes the buffer using the following flag bits:

- 0x01 Modified data
- 0x02 Data
- 0x04 LRU
- 0x08 Error

pagenum — is the physical page number on the disk.

memaddr — is the buffer memory address.

nslots — is the number of slot-table entries in the page. This field indicates the number of rows (or portions of a row) that are stored on the page.

pgflags — describes the page type using the following values, alone or in combination:

- 1 data page
- 2 tblspace page
- 4 free-list page

	8	chunk free-list page
	9	remainder data page
	b	partition resident blobpage
	c	blobspace resident blobpage
	d	blob chunk free-list bit page
	e	blob chunk blob map page
	10	B-tree node page
	20	B-tree root-node page
	40	B-tree branch-node page
	80	B-tree leaf-node page
	100	logical-log page
	200	last page of logical log
	400	sync page of logical log
	800	physical log
	1000	reserved root page
	2000	no physical log required
	8000	B-tree leaf with default flags
xflgs	describes buffer access using the following flag bits:	
	0x10	share lock
	0x80	exclusive lock
owner	is the user thread that set the xflgs buffer flag.	
waitlist	is the address of the first user thread waiting for access to this buffer. For a complete list of all threads waiting for the buffer, refer to "-X Option" on page 35-94.	

The number of modified buffers, the number of total buffers available, the number of hash buckets available, and the size of the buffer in bytes (the page size) are also listed. The maximum number of buffers available is specified as BUFFERS in the ONCONFIG file

The **-b** and **-B** options also provides summary information as shown in the following example:

```
123 modified, 23 resident, 2000 total, 2048 hash buckets, 2048 buffer size.
```

-d Option

Use the **-d** option to display information for chunks in each space. You can interpret output from this option as follows. The first section of the display describes the dbspaces:

address — is the address of the space in the shared-memory space table.

number — is the unique ID number of the space assigned at creation.

flags — describes each space using the following hexadecimal values:

0x0001	No mirror
0x0002	Mirror
0x0004	Down
0x0008	Newly mirrored
0x0010	Blobspace

fchunk — is the ID number of the first chunk.

nchunks — is the number of chunks in the space.

flags — describes each space using the following letter codes:

Position 1:	M -- Mirrored
	N -- Not Mirrored
Position 2:	X -- Newly mirrored
	P -- Physically recovered, waiting for logical recovery
	L -- Being logically recovered
	R-- Being recovered
Position 3:	B -- Blobspace

owner — is the owner of the space.

name — is the name of the space.

The line immediately following the space list includes the number of active spaces (the current number of dbspaces in the database server instance including the rootdbs) and the number of total spaces.

`Active spaces` refers to the current number of spaces in the database server instance including the rootdbs. `Total` refers to total *allowable* spaces for this database server instance.

The second section of the **onstat -d** output describes the chunks:

address	is the address of the chunk.
chk / dbs	is the chunk number and the associated space number.
offset	is the offset into the device in pages.
size	is the size of the chunk in pages.
free	is the number of free blobpages in the chunk. (A tilde indicates an approximate value for blobspaces.)
bpages	is the size of the chunk in blobpages. Blobpages can be larger than disk pages; therefore, the `bpages` value can be less than the `size` value.
flags	gives the chunk status information as follows: Position 1: P -- Primary M -- Mirror Position 2: O -- On-line D -- Down X -- Newly mirrored I -- Inconsistent Position 3: B -- Blobspace - -- Dbspace T - Temporary dbspace
pathname	is the pathname of the physical device.

The line immediately following the chunk list includes the number of active chunks (the number of chunks in the database server instance including the root chunk) and the number of total chunks.

Timing Affects onstat -d Output

Occasionally, the timing of the **onstat -d** command can affect the utility output. Timing becomes a factor in two cases. The first case occurs immediately after blobspace blobpages are allocated. The **onstat -d** routine looks directly at the disk to obtain blobpage statistics from the blobspace free-map page. If blobpages were recently allocated, **onstat -d** might not reflect the new allocation. This situation could arise if you execute **onstat -d** while the newest version of the blobspace free-map page remains in a memory buffer and is not yet flushed to disk.

Blobspace Page Types

The second case in which timing affects output occurs after blobspace blobpages are freed. The **onstat -d** output does not show a blobpage as free until the logical log in which the page or pages were deallocated is freed. That is, if you modify TEXT or BYTE data, **onstat -d** shows that the pages where the obsolete TEXT or BYTE data are stored are still in use until you back up and free the logical log that contains the modifying statement.

The **oncheck -pB** command, which examines the disk pages, does not reflect this timing nuance. If you delete TEXT or BYTE data from a blobspace, **oncheck -pB** output reflects the freed space immediately. For this reason, output from **onstat -d** and **oncheck -pB** could appear to be inconsistent.

For information about page reads and page writes, refer to **onstat -D**.

-D Option

Use the **-D** option to display page-read and page-write information for the first 50 chunks in each space. All but two of the fields that appear in the **-D** output also appear in the **onstat -d** output. You can interpret the two fields that are unique to the **-D** output as follows:

page Rd is the number of pages read.
page Wr is the number of pages written.

-f Option

Use the **-f** option 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 **onspaces**. When you execute **onstat -f**, the database server displays one of the following three outputs:

- Dataskip is OFF for all dbspaces.
- Dataskip is ON for all dbspaces.
- Dataskip is ON for dbspaces:

  dbspace1 dbspace2...

-F Option

Use the **-F** option to display a count for each type of write that flushes pages to disk. You can interpret output from this option as follows:

Fg Writes is the number of times that a foreground write occurred.

LRU Writes is the number of times that an LRU write occurred.

Chunk Writes is the number of times that a chunk write occurred.

address is the address of the user structure assigned to this page-cleaner thread.

flusher is the page-cleaner number.

state indicates the current page-cleaner activity using the following codes:

- C chunk write
- E exit
- I cleaner is idle
- L LRU queue

The exit code indicates either that the database server is performing a shutdown or that a page cleaner did not return from its write in a specific amount of time. This is also known as a time-out condition. Dynamic Server does not know what happened to the cleaner, so it is marked as exit. In either case, the cleaner thread eventually exits.

data provides additional information in concert with the **state** field. If **state** is C, **data** is the chunk number to which the page cleaner is writing buffers. If **state** is L, **data** is the LRU queue from which the page cleaner is writing. The **data** value is displayed as a decimal, followed by an equal sign, and repeated as a hexadecimal.

-g Monitoring Options

The following **onstat -g** options are provided for support and debugging only. You can include only one of these options per **onstat -g** command:

act	Prints active threads.
afr *pool name* | *session id*	Prints allocated memory fragments for a specified session or shared-memory pool. Each session is allocated a pool of shared memory. See **-mem** option for pool name.
all	Prints all multithreading information.
ath	Prints all threads. The **sqlmain** threads represent client sessions, and the *rstcb* value corresponds to the **user** field of the **onstat -u** command.
con	Prints conditions with waiters.
dic *table*	Without any parameters, this option will print one line of information for each table cached in the shared-memory dictionary. If given a specific table name as a parameter, it will print internal SQL information for that table.
dll	Prints a list of dynamic libraries that have been loaded.
dri	Prints data-replication information. (See "Monitoring High-Availability Data-Replication Status" on page 29-71 for a description of the information that is displayed.)
dsc	Prints data-distribution cache information.
ffr *pool name* | *session id*	Prints free fragments for a pool of shared memory.
glo	Prints global multithreading information. This information includes CPU use information about the virtual processors, the total number of sessions, and other multithreading global counters.
iob	Prints big-buffer use by I/O virtual processor class.
iof	Prints asynchronous I/O statistics by chunk/file. This option is similar to **-d** option, except that information on nonchunk files is also displayed. It includes information about temporary files and sort-work files.
iog	Prints asynchronous I/O global information.
ioq	Prints asynchronous I/O queuing statistics.
iov	Prints asynchronous I/O statistics by virtual processor.

lmx	Prints all locked mutexes.
mem *pool name* | *session id*	Prints memory statistics for a pool. Session pools are named with the session number. If no argument is provided, information about all pools is displayed.
mgm	Print MGM resource information.
nbm	Prints block bit map for the nonresident segments, one bit per 8-kilobyte block. Bit set indicates block free.
nsc *client id*	Prints shared-memory status by *client id*. If *client id* is omitted, all client status areas are displayed.

Tip: *This command prints the same status data as the* ***nss*** *command.*

nsd	Prints network shared-memory data for poll threads.
nss *session id*	Prints network shared- memory status by *session id*. If *session id* is omitted, all session status areas are displayed.

Tip: *This command prints the same status data as the* ***nsc*** *command.*

ntd	Prints network statistics by service.
ntm	Prints network mail statistics.
ntt	Prints network user times.
ntu	Prints network user statistics.
pos	Prints **.infos.DBSERVERNAME** file.
ppf *partition number* | 0	Prints partition profile for *partition number;* 0 prints profiles for all partitions.
prc	Prints stored procedure cache information.
qst	Prints queue statistics.
rbm	Prints block bit map for the resident segment (communication message area).
rea	Prints ready threads.
sch	For each virtual processor, this command prints the number of semaphore operations, spins, and busy waits.

seg	Prints shared-memory-segment statistics. This option shows how many segments are attached and their sizes.
ses *session id*	Prints session information by *session id*. If it is missing, a one-line summary of each session is printed.
sle	Prints all sleeping threads.
spi	Prints spin locks that virtual processors have spun more than 10,000 times to acquire. These spin locks are called longspins. The total number of longspins is printed in the heading of the **glo** command. Excessive longspins might indicate an overloaded system, too many virtual processors for a given computer, or an internal problem. To reduce longspins, reduce the number of (generally class CPU) virtual processors, reduce the load on the computer, or use the *no-age* or *processor* affinity features.
sql *session id*	Prints SQL information by *session id*. If *session id* is omitted, a one-line summary for each session is printed.
stk *tid*	Dumps stack of specified thread specified by thread ID. This option is not supported on all platforms and is not always accurate.
sts	Prints maximum and current stack use per thread.
tpf *tid*	Prints thread profile for *tid*; 0 prints profiles for all threads.
ufr *pool name* | *session id*	Prints allocated fragments by use.
wai	Prints waiting threads; all threads waiting on mutex or condition, or yielding.
wmx	Prints all mutexes with waiters.
wst	Prints wait statistics.

-i Option

Use the **-i** option to put **onstat** in interactive mode. In interactive mode, you can enter multiple **onstat** options per session, but only one at a time. An **onstat** prompt appears, allowing you to enter an option.

In interactive mode, do not precede the option with a dash.

Two additional options, **r** *seconds* and **rz** *seconds*, are available in interactive mode. The **r** *seconds* option is similar to the current **onstat -r** *seconds* option, which repeatedly generates a display. If an administrator executes **r** *seconds* at the interactive-mode prompt, the prompt changes to reflect the specified interval in seconds and reappears, waiting for the next command. In the following example, the display generated by the next command repeats every three seconds:

```
Informix Dynamic Server Version 7.30.UC1 -- On-Line -- Up 3
days 20:55:15 -- 4868 Kbytes

onstat> r 3
onstat[3]>
```

The **rz** *seconds* option enables you to repeat the next command as specified and to zero all profile counters between each execution.

Enter CTRL-D to terminate interactive mode.

Enter CTRL-C to terminate a repeating sequence.

-k Option

Use the **-k** option to display information about active locks. You can interpret output from this option as follows:

address	is the address of the lock in the lock table. If a user thread is waiting for this lock, the address of the lock appears in the `wait` field of the **onstat -u** (users) output.
wtlist	is the first entry in the list of user threads waiting for the lock, if there is one.
owner	is the shared-memory address of the thread holding the lock. This address corresponds to the address in the **address** field of **onstat -u** (users) output.

lklist	is the next lock in a linked list of locks held by the owner just listed.
type	uses the following codes to indicate the type of lock: HDR header B bytes lock S shared X exclusive I intent U update IX intent-exclusive IS intent-shared SIX shared, intent-exclusive
tblsnum	is the tblspace number of the locked resource.
rowid	is the row identification number. The rowid provides the following lock information: ■ If the rowid equals zero, the lock is a table lock. ■ If the rowid ends in two zeros, the lock is a page lock. ■ If the rowid is six digits or less and does not end in zero, the lock is probably a row lock. ■ If the rowid is more than six digits, the lock is probably an index key value lock.
key#/bsiz	is the index key number, or the number of bytes locked for a VARCHAR lock.

The maximum number of locks available is specified as LOCKS in the ONCONFIG file.

-l Option

Use the **-l** option to display information about physical and logical logs. You can interpret output from this option as follows. The first section of the display describes the physical-log configuration:

buffer	is the number of the physical-log buffer.
bufused	is the number of pages of the physical-log buffer that are used.
bufsize	is the size of each physical-log buffer in pages.
numpages	is the number of pages written to the physical log.

numwrits	is the number of writes to disk.
pages/io	is calculated as (numpages)/(numwrits). This value indicates how effectively physical-log writes are being buffered.
phybegin	is the physical page number of the beginning of the log.
physize	is the size of the physical log in pages.
phypos	is the current position in the log where the next log record write will occur.
phyused	is the number of pages used in the log.
%used	is the percent of pages used.

The second section of the **onstat -l** display describes the logical-log configuration:

buffer	is the number of the logical-log buffer.
bufused	is the number of pages used in the logical-log buffer.
bufsize	is the size of each logical-log buffer in pages.
numrecs	is the number of records written.
numpages	is the number of pages written.
numwrits	is the number of writes to the logical log.
recs/pages	is calculated as (numrecs/numpages). You cannot affect this value. Different types of operations generate different types (and sizes) of records.
pages/io	is calculated as (numpages/numwrits). You can affect this value by changing the size of the logical-log buffer (specified as LOGBUFF in the ONCONFIG file) or by changing the logging mode of the database (from buffered to unbuffered, or vice versa).

The following fields are repeated for each logical-log file:

address	is the address of the log-file descriptor.
number	is the logical-log file logid number.
flags	gives the status of each log as follows: A newly added B backed up C current logical-log file

	F free, available for use
	L contains the most-recent checkpoint record
	U used
uniqid	is the unique ID number of the log.
begin	is the beginning page of the log file.
size	is the size of the log in pages.
used	is the number of pages used.
%used	is the percent of pages used.

-m Option

Use the **-m** option to display the 20 most-recent lines of the system message log.

Output from this option lists the full pathname of the message-log file and the 20 file entries. A date-and-time header separates the entries for each day. A time stamp prefaces single entries within each day. The name of the message log is specified as MSGPATH in the ONCONFIG file.

-O Option

Use the **-O** option of the **onstat** utility to display information about the Optical Subsystem memory cache and staging-area blobspace. You can interpret output from this option as follows. 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**.

The first section of the display describes the following system-cache totals information:

size	is the size specified in the OPCACHEMAX configuration parameter.
alloc	is the number of 1-kilobyte pieces that the database server allocated to the cache.
avail	describes how much of **alloc** (in kilobytes) is not used.
number	is the number of TEXT or BYTE objects that the database server successfully put into the cache without overflowing.

kbytes	is the number of kilobytes of TEXT or BYTE data that the database server put into the cache without overflowing.
number	is the number of TEXT or BYTE objects that the database server wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of the 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 TEXT or BYTE object that has been processed. When the values in the **alloc** and **avail** output are equal to each other, the cache is empty.

The second section of the display describes the following user-cache totals information:

SID	is the session ID for the user.
use	is the user ID of the client.
size	is the size specified in the **INFORMIXOPCACHE** environment variable, if it is 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	is the number of TEXT or BYTE objects that the database server put into cache without overflowing.
kbytes	is the number of kilobytes of TEXT or BYTE data that the database server put into the cache without overflowing.
number	is the number of TEXT or BYTE objects that the database server wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of TEXT or BYTE data that the database server wrote to the staging-area blobspace.

-p Option

Use the **-p** option to display profile counts. The **-p** option displays profile counts either since you started the database server or since you ran **onstat** with the **-z** option.

The first portion of the display describes reads and writes.

Reads and writes are tabulated in three categories: from disk, from buffers, and number of pages (read or written).

The first **%cached** field is a measure of the number of reads from buffers compared to reads from disk. The second **%cached** field is a measure of the number of writes to buffers compared to writes to disk.

The database server buffers information and writes to the disk in pages. For this reason, the number of disk writes displayed as `dskwrits` is usually less than the number of writes executed by an individual user:

dskreads	is the number of actual reads from disk.
pagreads	is the number of pages read.
bufreads	is the number of reads from shared memory.
%cached	is the percent of reads cached, calculated as 100 * (bufreads - dskreads) / bufreads. (If dskreads exceeds bufreads, the value is displayed as `0.0`).
dskwrits	is the actual number of physical writes to disk. This number includes the writes for the physical and logical logs reported in **onstat -l**.
pagwrits	is the number of pages written.
bufwrits	is the number of writes to shared memory.
%cached	is the percent of writes cached, calculated as 100 *(bufwrits - dskwrits) / dskwrits). (If dskwrits exceeds bufwrits, the value is displayed as `0.0`).

The next portion of the **-p** display tabulates the number of times different ISAM calls were executed. The calls occur at the lowest level of operation and do not necessarily correspond one-to-one with SQL statement execution. A single query might generate multiple ISAM calls. These statistics are gathered across the database server and cannot be used to monitor activity on a single database unless only one database is active or only one database exists:

isamtot	is the total number of calls.
open	increments when a tblspace is opened.
start	increments when positioning within an index.
read	increments when the read function is called.
write	increments with each write call.
rewrite	increments when an update occurs.
delete	increments when a row is deleted.
commit	increments each time an **iscommit()** call is made. No one-to-one correspondence exists between this value and the number of explicit COMMIT WORK statements that are executed.
rollbk	increments when a transaction is rolled back.

The third portion of the **-p** display tracks the number of times that a resource was requested when none was available:

ovlock	is the number of times that the database server attempted to exceed the maximum number of locks (specified as LOCKS in the ONCONFIG file).
ovuser-threads	is the number of times that a user attempted to exceed the maximum number of user threads.
ovbuff	is the number of times that the database server could not find a free shared-memory buffers (specified as BUFFERS in the ONCONFIG file). When there are no free buffers, the database server writes out a dirty buffer to disk and then tries to find a free buffer.
usercpu	is the total user CPU time used by all user threads, expressed in seconds. This entry is updated once every 15 seconds.
syscpu	is the total system CPU time used by all user threads, expressed in seconds. This entry is updated once every 15 seconds.
numckpts	is the number of checkpoints since the boot time.
flushes	is the number of times that the buffer pool has been flushed to the disk.

The next portion of the **-p** display contains miscellaneous information, as follows:

bufwaits	increments each time that a user thread must wait for a buffer.
lokwaits	increments each time that a user thread must wait for a lock.
lockreqs	increments each time that a lock is requested.
deadlks	increments each time that a potential deadlock is detected and prevented.
dltouts	increments each time that the distributed deadlock time-out value is exceeded while a user thread is waiting for a lock.
ckpwaits	is the number of checkpoint waits.
compress	increments each time that a data page is compressed. (Refer to page compression.)
seqscans	increments for each sequential scan.

The last portion of the **-p** display contains the following information:

ixda-RA	is the count of read-aheads going from index leaves to data pages.
idx-RA	is the count of read-aheads traversing index leaves.
da-RA	is the count of data-path-only scans.
RA-pgsused	indicates the number of pages used that the database server read ahead. If this number is significantly less than the total number of pages read ahead, your read-ahead parameters might be set too high.
lchwaits	increments when a thread waits to gain access to a shared-memory resource.

-P Option

Use the **-P** option to display for all partitions the partition number and the break up of pages in the buffer pool that belong to the partition.

-R Option

Use the **-R** option to display detailed information about the LRU queues, FLRU queues, and MLRU queues. For an in-depth discussion of the three types of queues, see "LRU Queues" on page 11-37.

For each queue, the **onstat -R** lists the number of buffers in the queue and the number and percentage of buffers that have been modified. You can interpret output from this option as follows:

#	Queue number. Each LRU queue is composed of two subqueues: an FLRU queue and a MLRU queue. (See "LRU Queues" on page 11-37 for a definition of MLRU and FLRU queues.) Thus, queues 0 and 1 belong to the first LRU queue, queues 2 and 3 belong to the second LRU queue, and so on.
f / m	Identifies queue type. This field has four possible values: f free LRU queue. Free in this context means not modified. Although nearly all the buffers in an LRU queue are available for use, Dynamic Server attempts to use buffers from the FLRU queue rather than MLRU queue. (A modified buffer must be written to disk before Dynamic Server can use the buffer.) F free LRU with fewest elements. Dynamic Server uses this estimate to determine where to put unmodified (free) buffers next. m MLRU queue. M MLRU queue that is being cleaned by a flusher.
length	length of queue measured in buffers.
% of	percent of LRU queue that this subqueue composes. For example, suppose that an LRU queue has 50 buffers, with 30 of those buffers in the MLRU queue and 20 in the FLRU queue. The **% of** column would list percents of 60.00 and 40.00, respectively.
pair total	total number of buffers in this LRU queue.

Summary information follows the individual LRU queue information. You can interpret the summary information as follows:

dirty	is the total number of buffers that have been modified in all LRU queues.
queued	is the total number of buffers in LRU queues.
total	is the total number of buffers.
hash buckets	is the number of hash buckets.
buffer size	is the size of each buffer.
start clean	is the value of LRU_MAX_DIRTY.
stop at	is the value of LRU_MIN_DIRTY.

-s Option

Use the **-s** option to display general latch information. You can interpret output from this option as follows:

name — identifies the resource that the latch controls with the following abbreviations:

archive	dbspace backup
bf	buffers
bh	hash buffers
chunks	chunk table
ckpt	checkpoints
dbspace	dbspace table
flushctl	page-flusher control
flushr	page cleaners
locks	lock table
loglog	logical log
LRU	LRU queues
physb1	first physical-log buffer
physb2	second physical-log buffer
physlog	physical log
pt	tblspace tblspace
tblsps	tblspace table
users	user table

address	is the address of the latch. This address appears in the **-u** (users) output wait field if a thread is waiting for the latch.
lock	indicates if the latch is locked and set. The codes that indicate the lock status (1 or 0) are computer dependent.
wait	indicates if any user thread is waiting for the latch.
userthread	is the shared-memory address of the owner of the latch. In contrast to the user fields of other **onstat** options, this field does not contain the rstcb (RSAM task-control block) address because not all threads that own latches are user threads. Instead this field contains the thread-control block address, which all threads have. To obtain the rstcb address from the tcb address, examine the output of the **onstat -g ath** option, which lists both addresses for each user thread.

-t Option

Use the **-t** option to display tblspace information for active tblspaces, including whether tblspaces are memory-resident. You can interpret output from this option as follows:

n	is a counter of open tblspaces.
address	is the address of the tblspace in the shared-memory tblspace table.
flgs	describes the flag using the following flag bits: 0x01 Busy 0x02 Dirty (modified pages that have not been flushed to disk)
ucnt	is the usage count, which indicates the number of user threads currently accessing the tblspace.
tblnum	is the tblspace number expressed as a hexadecimal value. The integer equivalent appears as the **partnum** value in the **systables** system catalog table.
physaddr	is the physical address (on disk) of the tblspace.
npages	is the number of pages allocated to the tblspace.
nused	is the number of used pages in the tblspace.
npdata	is the number of data pages used.

nrows	is the number of data rows used.
nextns	is the number of (noncontiguous) extents allocated. This number is not the same as the number of times that a next extent has been allocated.
resident	indicates whether tblspace is memory-resident; 1 = yes, 0= no.

The number of active tblspaces, the number of available tblspaces, and the number of available hash buckets are also listed.

-u Option

Use the **-u** option to print a profile of user activity. The output described here is provided for each user thread.

You can interpret output from **onstat -u** as follows:

address	is the shared-memory address of the user thread (in the user table). Compare this address with the addresses displayed in the **-s** output (latches); the **-b**, **-B**, and **-X** output (buffers); and the **-k** output (locks) to learn what resources this thread is holding or waiting for.
flags	gives the status of the session. The flag codes for position 1: B waiting on a buffer C waiting on a checkpoint G waiting on a write of the logical-log buffer L waiting on a lock S waiting on mutex T waiting on a transaction Y waiting on condition X waiting on a transaction cleanup (rollback) The flag codes for position 2: * transaction active during an I/O failure The flag codes for position 3: A dbspace backup thread See the third position of flag codes for **-x** option for other values that appear here.

The flag codes for position 4:

P primary thread for a session

The flag codes for position 5:

R Reading (RSAM call)

X Thread in critical section.

The flag codes for position 7:

B btree cleaner thread

C terminated user thread waiting for cleanup

D a daemon thread

F a page-cleaner thread

M special ON-Monitor (monitor) thread

sessid	is the session identification number. During operations such as parallel sorting and parallel index building, a session might have many user threads associated with it. For this reason, the session ID identifies each unique session.
user	is the user login name (derived from UNIX).
tty	indicates the tty that the user is using (derived from UNIX).
wait	if the user thread is waiting for a specific latch or lock, this field displays the address of the resource. Use this address to map to information provided in the **-s** (latch) or **-k** (lock) output.
tout	is the number of seconds left in the current wait. If the value is `0`, the user thread is not waiting for a latch or lock. If the value is `-1`, the user thread is in an indefinite wait.
locks	is the number of locks that the user thread is holding. (The **-k** output should include a listing for each lock held.)
nreads	is the number of disk reads that the user thread has executed.
nwrites	is the number of write calls that the user thread has executed. All write calls are writes to the shared-memory buffer cache.

Figure 35-2 shows output from **onstat -u**. The last line of **onstat -u** output displays the maximum number of concurrent user threads that were allocated since you initialized the database server. For example, the last line of a sample **onstat -u** output is as follows:

```
4 active, 128 total, 17 maximum concurrent
```

The last part of the line, `17 maximum concurrent`, indicates that the maximum number of user threads that were running concurrently since you initialized the database server was `17`.

```
Informix Dynamic Server Version 7.30.UC1 -- On-Line -- Up 00:50:22 -- 8896 Kbytes

Userthreads
address  flags   sessid   user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
4 active, 128 total, 17 maximum concurrent
```

Figure 35-2
Output from onstat -u

The number of active users and the maximum number of users allowed are also indicated.

-x Option

The transaction information is required only for an X/Open environment or in some situations in which the database server is participating in queries managed by INFORMIX-STAR.

You can interpret output from **onstat -x** as follows:

address — is the shared-memory address of the transaction structure.

flags — The flag codes for position 1:

- A — User thread attached to the transaction
- S — INFORMIX-TP/XA suspended transaction
- C — INFORMIX-TP/XA waiting for rollback

The flag codes for position 3:

- B — Begin work
- P — INFORMIX-STAR prepared for commit
- X — INFORMIX-TP/XA prepared for commit
- C — Committing or committed

R Rolling back or rolled back

H Heuristically rolling back or rolled back

The flag codes for position 5:

G Global transaction

C INFORMIX-STAR coordinator

S INFORMIX-STAR subordinate

B Both INFORMIX-STAR coordinator and INFORMIX-STAR subordinate

userthread	is the user that owns the transaction (rstcb address).
locks	is the number of locks held by transaction.
log begin	is the log in which begin work was logged.
isolation	is the isolation level.
retrys	is the attempts to start an INFORMIX-STAR recovery thread.
coordinator	is the transaction coordinator when this transaction is a subordinate.

Figure 35-3 shows output from **onstat -x**. The last line of the **onstat -x** output displays the maximum number of concurrent transactions since you initialized the database server. For example, the last line of a **onstat -u** output is as follows:

```
11 active, 128 total, 6 maximum concurrent
```

The last part of the line, `6 maximum concurrent`, indicates that the maximum number of transactions that were running concurrently since you initialized the database server was 6.

```
Informix Dynamic Server Version 7.30.UC1-- On-Line -- Up 00:50:22 -- 8896 Kbytes

Transactions
address  flags userthread locks log begin isolation retrys coordinator
40a7e4   A---- 406464     0     0         COMMIT    0
40a938   A---- 4067c4     0     0         COMMIT    0
40aa8c   A---- 406b24     0     0         COMMIT    0
40abe0   A---- 40a124     0     0         COMMIT    0
11 active, 128 total, 6 maximum concurrent
```

Figure 35-3
Output from onstat -x

-X Option

Use the **-X** option to obtain precise information about the threads that are sharing and waiting for buffers.

For each buffer in use, the **-X** option displays general buffer information that is also available with either the **-b** or **-B** option. For an explanation of these fields, refer to **onstat -b**.

Unique to the **-X** option are the **sharers** and **waiters** fields. More than one thread can share data in a buffer. For each buffer, the **sharers** field lists the addresses of all user threads sharing that buffer. During an update operation, a thread places an exclusive lock on a buffer, and no sharing occurs. In this situation, the **waiters** field lists the addresses of all user threads that are waiting for the buffer.

The **onstat -b** and **-B** options contain a **waitlist** field that displays the address of the first user thread that is waiting for the buffer. The **-X** option provides a complete list of addresses for all waiting threads.

The maximum number of shared buffers is specified as BUFFERS in the ONCONFIG file.

-z Option

Use the **-z** option to set the profile counts to zero.

If you use the **-z** option to reset and monitor the count of some fields, be aware that profile counts are incremented for all activity that occurs in any database that Dynamic Server manages. Any user can reset the profile counts and thus interfere with monitoring that another user is conducting.

ontape: Logging, Archives, and Restore

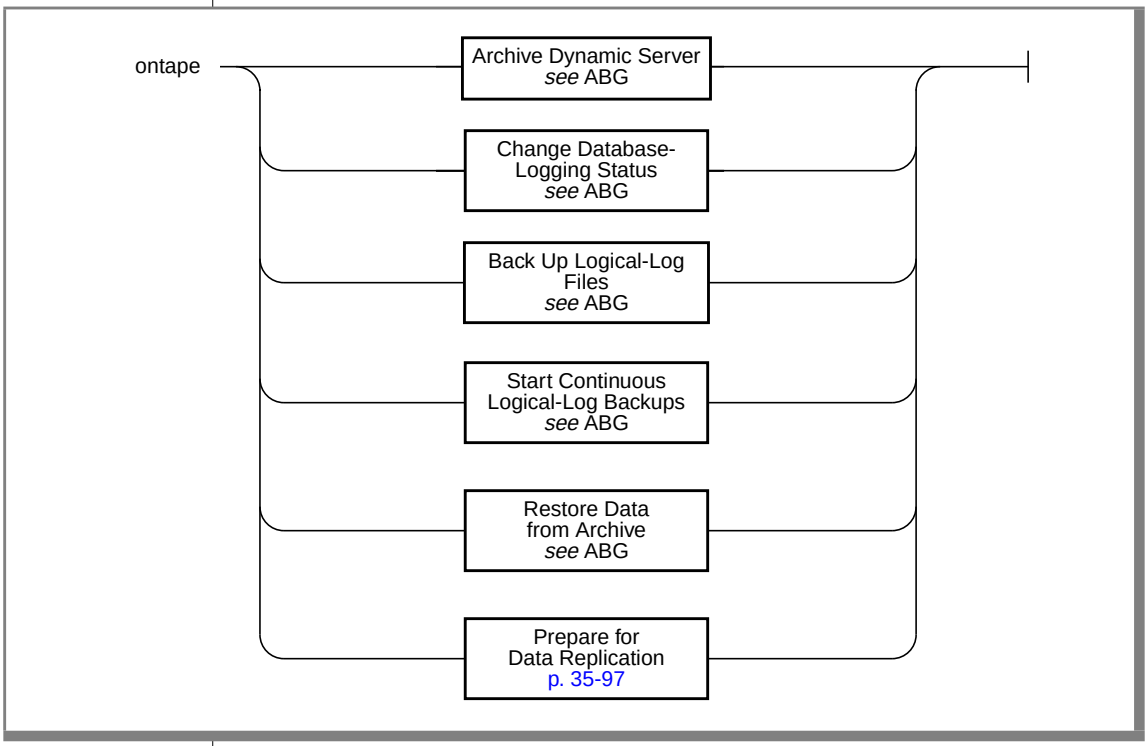
The **ontape** utility lets you perform several different tasks. Alternatively, you can use ON-Bar or ON-Archive to perform these tasks. For information about ON-Bar, see the *Backup and Restore Guide*. For information about ON-Archive, see the *Archive and Backup Guide*.

The **ontape** utility lets you perform the following tasks:

- Archive data that the database server manages
- Change database-logging status
- Back up logical-log files
- Start continuous logical-log file backups
- Restore data from an archive tape
- Use data replication

You must be logged in as user **root** or user **informix** to execute **ontape**.

Syntax



ABG refers to your *Archive and Backup Guide*. Syntax for **ontape** options other than **-t** and **-l** appear in that manual.

Things to Consider

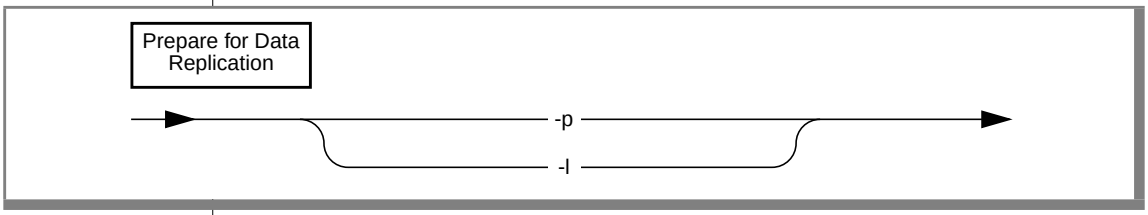
If more than one tape is needed during data replication, **ontape** prompts for each additional tape. Do not run **ontape** in background mode because you might need to provide input from the terminal or window.

Exit Codes

The **ontape** utility has two exit codes:

0 indicates a normal exit from **ontape.**
1 indicates an exceptional condition.

Prepare for Data Replication



Element	Purpose	Key Considerations
-l	Directs **ontape** to perform a logical restore on all the dbspaces that have just been physically restored on the database server in a data-replication pair.	**Additional Information**: This option rolls forward logical-log records from the last checkpoint up to the last available logical-log record on disk.
-p	Directs **ontape** to perform a physical restore of a database server.	**Additional Information**: This option is used expressly for replicating data prior to initiating the data-replication feature.

Use the **-p** and **-l** options to replicate data initially in a pair of database servers that use data replication. See "Starting High-Availability Data Replication for the First Time" on page 26-11.

Chapter

36

Message-Log Messages

How the Messages Are Ordered in This Chapter 36-3

Message Categories 36-4

Messages: A-B 36-4

Messages: C 36-6

Messages: D-E-F 36-14

Messages: G-H-I 36-19

Messages: J-K-L-M 36-20

Messages: N-O-P 36-26

Messages: Q-R-S 36-33

Messages: T-U-V 36-38

Messages: W-X-Y-Z 36-42

Messages: Symbols 36-43

This chapter contains nonnumbered messages that are printed in the Informix Dynamic Server message log. It lists the nonnumbered messages that can appear in the message log and provides explanatory notes for the messages. If numbered messages appear, see *Informix Error Messages* in Answers OnLine to look up their explanations and corrective actions.

Of the messages included here, a few might require you to contact Informix Technical Support staff. Such messages are rarely, if ever, seen at customer locations.

For information on what the message log is, the location of the file where Dynamic Server sends the messages, and some guidance on how and when you might want to read it, see "What Is the Message Log?" on page 29-7.

For error messages that pertain to ON-Bar, see your *Backup and Restore Guide*. For error messages that pertain to ON-Archive, see your *Archive and Backup Guide*.

How the Messages Are Ordered in This Chapter

Database server message-log messages are arranged in this chapter in alphabetical order, sorted with the following additional rules:

- The time stamp that precedes each message is ignored.
- Letter case is ignored in alphabetization.
- Spaces are ignored.
- Quotation marks are ignored.
- The word *the* is ignored if it is the first word in the message.
- Messages that begin with numbers or punctuation symbols appear toward the end of the list in a special section labeled "Messages: Symbols" on page 36-43.

A cause and suggested corrective action for a message or group of messages follows the message text.

Message Categories

Four general categories of messages can be defined, although some messages fall into more than one category:

- Routine information
- Assertion-failed messages
- Administrative action needed
- Fatal error detected

The assertion-failed messages reflect their traditional use by Informix technical staff to assist in troubleshooting and diagnostics. The information that they report often falls into the category of *unexpected events* that might or might not develop into problems caught by other error codes. Moreover, the messages are terse and often extremely technical. They might report on one or two isolated statistics without providing an overall picture of what is happening.

When technical staff are investigating a problem, this information can suggest to them possible research paths. However, you might find that the information has little or no application when it is taken out of this context or when processing is proceeding normally.

Messages: A-B

`Aborting Long Transaction: tx` *`0xn`*

Cause: The transaction spans the log space specified by transaction high-water mark (LTXHWM), and the offending long transaction is rolling back.

Action: No additional action is needed. The address of the transaction structure in shared memory is displayed as a hexadecimal value.

```
Affinitied VP mm to phys proc nn
```

Cause: The database server successfully bound a CPU virtual processor to a physical processor.

Action: None required.

```
Affinity not enabled for this server
```

Cause: You tried to bind your CPU virtual processors to physical processors, but the database server that you are running does not support process affinity.

Action: Set AFF_NPROCS to zero, or remove the affinity setting from VPCLASS.

```
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 administrator should take
See Also: DUMPDIR/af.uniqid containing more diagnostics.
```

Cause: This message indicates an internal error.

Action: The **af.*uniqid*** file in the directory specified by the ONCONFIG parameter DUMPDIR 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 / or data buffers. The information included in this message is intended for Informix Technical Support.

Contact Informix Technical Support.

```
Begin re-creating indexes deferred during recovery.
```

Cause: During recovery, indexes to be created are deferred until after recovery completes. This message indicates that the database server deferred re-creating indexes and that it is now creating the indexes. During the time that the database server re-creates the indexes, it locks the affected tables with a shared lock.

Action: None required.

```
Beginning process of reverting to 5.0 disk structure...
```

Cause: The database server started reverting your disk space to Version 5.0 format.

Action: None required.

```
Building 'sysmaster' database requires ~mm pages of logical log.
Currently there are nn pages available.
Prepare to back up your logs soon.
```

Cause: You do not currently have the approximate amount of free log space necessary to complete a build of the **sysmaster** database.

Action: Back up your logs.

```
Building 'sysmaster' database...
```

Cause: The database server is building the **sysmaster** database.

Action: None required.

Messages: C

```
Cannot Allocate Physical-log File, mm wanted, nn available
```

Cause: The database server attempted to initialize shared memory with a physical-log size that exceeds the amount of contiguous space available in the dbspace (specified as PHYSDBS in ONCONFIG). Both quantities of space, wanted and available, are expressed as kilobytes.

Action: You must either reduce the size of the physical log (specified as PHYSFILE in ONCONFIG) or change the location of the physical log to a dbspace that contains adequate contiguous space to accommodate the physical log.

```
Cannot change to mode.
```

Cause: Some error during fast or full recovery has prevented the system from changing to on-line or quiescent mode.

Action: See previous messages in the log file for information, or contact Informix Technical Support.

```
Cannot Commit Partially Complete Transactions
```

Cause: Transactions that drop tables or indexes do not perform the drop until a COMMIT statement is processed (with a few exceptions). In these cases, a *beginning commit* log record is written, followed by the usual commit log record. If the database server fails in between the two, the fast recovery process will attempt to complete the commit the next time that you initialize the database server.

If this completion of the commit fails, the database server generates the preceding message.

Action: Examine the logical log as described in Chapter 37, "Interpreting Logical-Log Records," to determine if you need to take action.

```
Cannot create a user defined VP class with 'SINGLE_CPU_VP' non-
zero
```

Cause: SINGLE_CPU_VP is set to nonzero, and **onmode** was used to create a user-defined VP class.

Action: If user-defined VP classes are necessary, stop the database server, change SINGLE_CPU_VP to zero, and restart the database server.

Cannot Open Dbspace *nnn*

Cause: The database server is unable to access the specified dbspace. This message indicates a problem opening the tblspace tblspace or corruption in the initial chunk of the dbspace.

Action: Verify that the device or devices that make up the chunks of this dbspace are functioning properly and that you assigned them the correct operating-system permissions (rw-rw----). You might be required to perform a data restore.

Cannot Open Logical Log

Cause: The database server is unable to access the logical-log files. Because the database server cannot operate without access to the logical log, you must resolve this problem.

Action: Verify that the chunk device where the logical-log files reside is functioning and has the correct operating-system permissions (rw-rw----).

Cannot Open Mirror Chunk *pathname*, errorno = *nn*

Cause: The database server cannot open the mirrored chunk of a mirrored pair. The chunk pathname and the operating-system error are returned.

Action: See your operating-system documentation for more information about corrective actions.

Cannot Open Primary Chunk *pathname*, errorno = *nnn*

Cause: The primary chunk of a mirrored pair cannot be opened. The chunk pathname and the operating-system error are returned.

Action: See your operating-system documentation for more information about corrective actions.

Cannot Open Primary Chunk *chunkname*

Cause: The *initial* chunk of the dbspace cannot be opened.

Action: Verify that the chunk device is running properly and has the correct operating-system permissions (rw-rw----).

Cannot Perform Checkpoint, shut system down.

Cause: A thread that is attempting to restore a mirrored chunk has requested a checkpoint, but the checkpoint cannot be performed.

Action: Shut down the database server.

Cannot Restore to Checkpoint

Cause: The database server is unable to recover the physical log and thus unable to perform fast recovery.

Action: If the database server does not come on-line, perform a data restore from dbspace backup.

Cannot Rollback Incomplete Transactions

Cause: Within the fast-recovery or data-restore procedure, the logical-log records are first rolled forward. Then, open transactions that have not committed are rolled back. An open transaction could fail during the rollback, leaving some of the modifications from the open transaction in place. This error does not prevent the database server from moving to quiescent or on-line mode, but it might indicate an inconsistent database.

Action: Examine the logical log using the **onlog** utility to determine if any action is needed.

Cannot update pagezero

Cause: A failure occurred while the database server was trying to rewrite a reserved page during the reversion process.

Action: See previous messages in the log file for information, or call Informix Technical Support.

Can't affinity VP *mm* to phys proc *nn*

Cause: The database server supports process affinity, but the system call to bind the virtual processor to a physical processor failed.

Action: See your operating-system documentation.

shutdown

Cause: A dbspace has gone down during a checkpoint interval. The database server is configured to wait for an override when this situation occurs.

Action: Either shutdown the database server or issue an **onmode -O** command to override the down dbspace. For more information on the **onmode** utility, see "onmode: Mode and Shared-Memory Changes" on page 35-28

Checkpoint Completed: duration was *n* seconds

Cause: A checkpoint completed successfully.

Action: None required.

Checkpoint Page Write Error

Cause: The database server detected an error in an attempt to write checkpoint information to disk.

Action: Please contact Informix Technical Support for additional assistance in resolving this situation.

Checkpoint Record Not Found in Logical Log

Cause: The logical log or the chunk that contains the logical log is corrupted. The database server cannot initialize.

Action: Perform a data restore from dbspace backup.

Chunk number *nn pathname* -- Offline

Cause: The indicated chunk in a mirrored pair has been marked with status D and taken off-line. The other chunk in the mirrored pair is operating successfully.

Action: Take steps now to repair the chunk device and restore the chunk. The chunk number and chunk device pathname are displayed.

Chunk number *nn pathname* -- Online

Cause: The indicated chunk in a mirrored pair has been recovered and is on-line (marked with status O). The chunk number and chunk device pathname are displayed.

Action: None required.

Completed automatic conversion of system catalog indexes. Run UPDATE STATISTICS to automatically convert other indexes or drop and recreate them manually.

Cause: The index-conversion process was completed successfully from a Version 5.0 database.

Action: Run UPDATE STATISTICS to convert other indexes automatically, or drop and re-create them manually.

The chunk *pathname* must have READ/WRITE permissions for owner and group.

Cause: The chunk *pathname* does not have the correct owner and group permissions.

Action: Make sure that you assigned the correct permissions (-rw-rw---) to the device on which the chunk is located.

The chunk *pathname* must have owner-ID and group-ID set to informix.

Cause: The chunk *chunkname* does not have the correct owner and group ID.

Action: Make sure the device on which the chunk is located has the owner ID and group ID set to **informix**.

The chunk *pathname* will not fit in the space specified.

Cause: The chunk *pathname* will not fit in the space that you specified.

Action: Choose a smaller size for the chunk, or free space where the chunk is to be created.

The current number of *primary*/*mirror* chunks, *nn*, exceeds the Version 5.0 limit.

Cause: When you revert to Version 5.0, the number of chunks in existence must be within the Version 5.0 limit. Chunk number nn exceeds the Version 5.0 limit. This conflict will cancel the reversion process.

Action: Meet the preceding Version 5.0 limit. Use **onspaces** or ON-Monitor to drop enough chunks to meet the Version 5.0 limit.

Completed recreating indexes.

Cause: The database server finished recreating the deferred indexes.

Action: None required.

Completed partnum conversion

Cause: The database server successfully completed the partnum conversion process from a Version 5.0 database.

Action: None required.

Continuing Long Transaction (for COMMIT): tx *0xn*

Cause: The logical log has filled beyond the long-transaction high-water mark (LTXHWM), but the offending long transaction is in the process of committing. In this case, the transaction is permitted to continue writing to the logical log and is not rolled back. The address of the transaction structure in shared memory is displayed as hexadecimal value tx 0xn.

Action: None required.

`Converting indexes to 6.0 format...`

Cause: The database server started converting indexes to Version 6.0 format.

Action: None required.

`Converting partnums to 6.0 format...`

Cause: The database server started converting partnums to Version 6.0 format.

Action: None required.

`Could not disable priority aging: errno = nn`

Cause: While trying to disable priority aging for the CPU virtual processor, a UNIX system call failed. The system error number associated with the failure is returned.

Action: See your operating-system documentation.

`Could not fork a virtual processor: errno = nn`

Cause: The fork of a virtual processor failed. The database server returns the UNIX system error number associated with the failure.

Action: See your operating-system documentation. Check your operating-system kernel for the maximum number of processes available per user and for the system.

`Create_vp: cannot allocate memory`

Cause: The database server cannot allocate new shared memory.

Action: The database-server administrator must make more shared memory available. This situation might require increasing SHMTOTAL or reconfiguring the operating system. This message is usually accompanied by other messages that give additional information.

Messages: D-E-F

dataskip is OFF for all dbspaces

Cause: Informational.
Action: None Required.

dataskip is ON for all dbspaces

Cause: Informational.
Action: None required.

dataskip is ON for dbspaces: <*dbspacelist*>.

Cause: Informational; DATASKIP is ON for the specified dbspaces.
Action: None required.

dataskip will be turned {ON|OFF} for <*dbspacename*>.

Cause: Informational; DATASKIP is ON or OFF for the specified dbspace.
Action: None required.

DBSPACETEMP internal list not initialized, using default

Cause: An error occurred while initializing a user-specified DBSPACETEMP list. Typically this condition is due to a memory-allocation failure.
Action: Check for accompanying error messages.

The DBspace/BLOBspace *spacename* is now mirrored.

Cause: You successfully added mirroring to the indicated dbspace or blobspace.
Action: None required.

The DBspace/BLOBspace *spacename* is no longer mirrored.

Cause: You have ended mirroring for the indicated dbspace or blobspace.

Action: None required.

Dbspace *dbspacename* for Physical-log File not found

Cause: The dbspace *dbspacename* specified by the PHYSDBS configuration parameter does not exist. As a consequence, the database server cannot complete initialization.

Action: Use a dbspace known to exist.

Dbspace *dbspacename*, number *mm*, exceeds the Version 5.0 limit of *nn*.

Cause: When you revert to Version 5.0, the number of dbspaces in existence must be within the Version 5.0 limit. The dbspace named, with dbspace number *mm*, exceeds the Version 5.0 limit of *nn*. This conflict will cancel the reversion process.

Action: Drop dbspaces until the number meets the Version 5.0 limit.

devname: write failed, file system is full.

Cause: The file system *devname* being full causes write to fail.

Action: Free some space in *devname*.

`Dropping temporary tblspace 0xn, recovering nn pages.`

Cause: During shared-memory initialization, the database server routinely searches for temporary tables that are left without proper cleanup or following an uncontrolled shutdown. If a temporary table is found, the database server drops the table and recovers the space. The database server located the specified temporary tblspace and dropped it. The value *0xn* is the hexadecimal representation of the tblspace number.

Action: None required.

`dynamically allocated new shared memory segment (size nnnn)`

Cause: This status message informs you that the database server successfully allocated a new shared-memory segment of size *nnnn*.

Action: None required.

`ERROR: NO "waitfor" locks in Critical Section.`

Cause: The database server does not permit a thread to own locks that might have to wait while that thread is within a critical section. Any such lock request is denied, and an ISAM error message is returned to the user.

Action: The error reported is an internal error. Contact Informix Technical Support.

`Error building sysmaster database See outfile`

Cause: Errors were encountered in building the **sysmaster** database. The file *outfile* contains the result of running the script **buildsmi**.

Action: See the file *outfile*.

```
Error writing pathname errno = nn
```

Cause: Cannot write to *pathname*. The number *nn* is the number of the UNIX error returned.

Action: Investigate the cause of the UNIX error. Usually it means that no space is available for the file. It might also mean that the directory does not exist or that no write permissions exist.

```
Error writing shmem to file filename (error)
Unable to create output file filename errno=mm
Error writing filename errno=nn
```

Cause: The database server detected an error in an attempt to write shared memory to *filename*. The first message is followed by one of the next two. Either the attempt failed because the output file could not be created or because the contents of shared memory could not be written. The error refers to the UNIX operating-system error that prompted the attempted write of shared memory to a file. The value of *nn* is the operating-system error.

Action: See your operating-system documentation.

```
Existing sysmaster database renamed to sysmaster_pre60
```

Cause: When the database server was building the **sysmaster** database, it found that a database with the name of **sysmaster** already exists. The database server renamed the database **sysmaster_pre60**.

Action: None required.

```
Fatal error initializing CWD string.
Check permissions on current working directory. Group groupname
must have at least execute permission on '.'.
```

Cause: Group *groupname* does not have execute permission for the current working directory.

Action: Check permissions on the current working directory. You or the system administrator must give your group *execute* permission on the current working directory. Retry the operation that generated this message, after your group has been given permission.

```
Fragments dbspacename1 dbspacename2 of table tablename set to
non-resident.
```

Cause: The specified fragments of *tablename* either have been set to nonresident by the SET TABLE command.

Action: None required.

```
forced-resident shared memory not available
```

Cause: The database server port for your computer does not support forced-resident shared memory.

Action: None required.

```
Freed mm shared-memory segment(s) nn bytes
```

Cause: The database server sends this message to the message log after you run the **-F** option of the **onmode** utility to free unused memory. The message informs you of the number of segments and bytes that the database server successfully freed.

Action: None required.

Messages: G-H-I

`gcore pid; mv core.pid dir/core.pid.ABORT`

Cause: This status message during a database server failure provides the name and place of each core file associated with the virtual processors.

Action: None required.

`I/O function chunk mm, pagenum nn, pagecnt aa --> errno = bb`

Cause: An operating-system error occurred during an attempt to access data from disk space. The operating-system function that failed is defined by *function*. The chunk number and physical address of the page where the error occurred are displayed as integers.The *pagecnt* value refers to the number of pages that the thread was attempting to read or write. If an *errno* value is displayed, it is the number of the operating-system error and might explain the failure. If *function* is specified as *bad request*, some unexpected event caused the I/O attempt on an invalid chunk or page.

Action: If the chunk status changes to D, or down, restore the chunk from its mirror, or repair the chunk. Otherwise, perform a data restore.

`I/O error, primary/mirror Chunk pathname -- Offline (sanity)`

Cause: The database server detected an I/O error on a primary or mirrored chunk with *pathname*. The chunk was taken off-line.

Action: Check that the device on which the chunk was stored is functioning as intended.

`Indexes idx1 and idx2 set to non-resident`

Cause: The specified indexes have been set to nonresident through the SET TABLE statement.

Action: None required.

Informix Dynamic Server Initialized - Complete Disk Initialized

Cause: Disk space and shared memory have been initialized. Any databases that existed on the disk before the initialization are now inaccessible.

Action: None required.

Informix Dynamic Server Initialized - Shared Memory Initialized

Cause: Shared memory has been initialized.

Action: None required.

Informix Dynamic Server Stopped

Cause: The database server has moved from quiescent mode to off-line mode. The database server is off-line.

Action: None required.

Internal overflow of shmid's, increase system max shared memory segment size.

Cause: The database server was initializing shared memory when it ran out of internal storage for the shmids associated with this segment.

Action: Increase the value of your maximum kernel shared-memory segment size, usually SHMMAX. See your operating-system documentation for more information.

Messages: J-K-L-M

Listener-thread err = *error_number*: *error_message*

Cause: A listener thread has encountered an error. This message displays the error number and message text.

Action: See the *Informix Error Messages* manual for the cause and corrective action.

`Lock table overflow - user id mm session id nn`

Cause: A thread attempted to acquire a lock when no locks were available. The user ID and session ID are displayed.

Action: Increase the LOCKS configuration parameter, and initialize shared memory.

`Logical-log File not found`

Cause: The checkpoint record in the root dbspace reserved page is corrupted.

Action: Perform a data restore from dbspace backup.

`Logical Log nn Complete`

Cause: The logical-log file identified by log-ID number *nn* is full. The database server automatically switches to the next logical-log file in the sequence.

Action: None required.

`Logical logging error for type:subtype in (????)`

Description: The log record that caused the error is identified as follows:

type	is the logical-log record type.
subtype	is the logging subsystem (OLDRSAM, RSAM).
????	is the name of an internal function that indicates what system failed to log.

Cause: Logging failure.

Action: Contact Informix Technical Support.

`Log number nn, exceeds the Version 5.0 limit of mm.`

Cause: When you revert to Version 5.0, the number of logs in existence must be within the Version 5.0 limit. Log number *nn* exceeds the Version 5.0 limit of *mm*. This conflict will cancel the reversion process.

Action: Meet the Version 5.0 limit.

```
Log Record: log = ll, pos = 0xn, type = type:subtype(snum),
trans = xx
```

Description: The log record that caused the error is identified as follows:

ll	is the logical-log log ID where the record is stored.
0xn	is the hexadecimal address position within the log.
type	is the logical-log record type.
subtype	is the logging subsystem (OLDRSAM, RSAM).
snum	is the subsystem number.
xx	is the transaction number that appears in the logical log.

Cause: The database server detected an error during the rollforward portion of fast recovery or logical-log restore.

Action: Contact Informix Technical Support.

```
Log record (type:subtype) at log nn, 0xn was not undone
```

Description: The log record that caused the error is identified as follows:

type	is the logical-log record type.
subtype	is the logging subsystem (OLDRSAM, RSAM).
nn	is the logical-log log ID where the record is stored.
0xn	is the hexadecimal address position within the log.

Cause: A log undo failed because a log is corrupt.

Action: Examine the logical log using the **onlog** utility to determine if any action is needed. Contact Informix Technical Support.

`Log record (type:subtype) failed, partnum pnum row rid iserrno num`

Description: The log record that caused the error is identified as follows:

type	is the logical-log record type.
subtype	is the logging subsystem (OLDRSAM, RSAM).
pnum	is the part number.
rid	is the row id.
num	is the iserror number.

Cause: A logging failure occurred.

Action: Contact Informix Technical Support.

`Log record (type:subtype) in log nn, offset 0xn was not rolled back`

Description: The log record that caused the error is identified as follows:

type	is the logical-log record type.
subtype	is the logging subsystem (OLDRSAM, RSAM).
log	is the logical-log log ID where the record is stored.
offset	is the hexadecimal address position within the log.

Cause: A log undo failed because a log is corrupt.

Action: Examine the logical log using the **onlog** utility to determine if any action is needed. Contact Informix Technical Support.

Logical Recovery allocating *nn* worker threads *thread_type*.

Cause: The database server determined the number of worker threads that will be used for parallel recovery. The variable *thread_type* can assume the values ON_RECVRY_THREADS or OFF_RECVRY_THREADS.

Action: This message is a status message. No action is required. If you want a different number of worker threads allocated for parallel recovery, change the value of the ONCONFIG configuration parameter ON_RECVRY_THREADS or OFF_RECVRY_THREADS.

Logical Recovery Started

Cause: Logical recovery began.

Action: This message is a status message. No action is required.

Memory allocation error.

Cause: This error could be caused by either not enough main (OS) memory or not enough internal (database server) memory.

Action: See your operating-system documentation for information on how to increase OS memory. Alternatively, increase the virtual-memory size (SHMVIRTSIZE), the size of the added segments, (SHMADD) or your total shared-memory size (SHMTOTAL).

Mirror Chunk *chunkname* added to space *spacename*. Perform manual recovery.

Cause: Fast recovery, full recovery, or an HDR secondary has recovered the add of a mirror chunk. It does not perform automatic mirror recovery, however. The administrator must do this.

Action: Attempt to recover the mirror chunks using either the **onspaces** utility or ON-Monitor.

`Mixed transaction result. (pid=`*nn* `user=`*userid*`)`

Cause: You receive this message only when more than one database server is involved in a transaction. This message indicates that a database server, after preparing a transaction for commit, heuristically rolled back the transaction, and the global transaction completed inconsistently. The *pid* value is the user-process identification number of the coordinator process. The value of *user* is the user ID associated with the coordinator process.

Action: See Chapter 31, "Recovering Manually from Failed Two-Phase Commit."

`mt_shm_free_pool: pool` *0xn* `has blocks still used (id` *nn*`)`

Cause: An internal error occurred during a pool deallocation because blocks are still associated with the pool.

Action: Contact Informix Technical Support.

`mt_shm_init: can't create` *resident/virtual* `segment`

Cause: The causes for the failure to create the resident or virtual segment are as follows: (1) the segment size is less than the minimum segment size; (2) the segment size is larger than the maximum segment size; (3) allocating another segment would exceed the allowable total shared-memory size; or (4) a failure occurred while trying to allocate the segment.

Action: If you suspect that this error was generated because of item 1 or 2 in the preceding paragraph, contact Informix Technical Support. To correct item 3, increase the SHMTOTAL value in your ONCONFIG configuration file. See your logical-log file for additional information about errors generated because of item 4.

```
mt_shm_remove: WARNING: may not have removed all/correct
segments
```

Cause: While removing the shared-memory segments associated with the database server, the last segment removed does not equal the last segment registered internally. This situation is probably due to the unexpected failure of the database server.

Action: Remove any segments that were not cleaned up.

Messages: N-O-P

```
Not enough main memory
```

Cause: The database server detected an error in an attempt to acquire more memory space from the operating system.

Action: See your operating-system documentation for more information about **shmget()**.

```
Not enough Logical-log files, Increase LOGFILES
```

Cause: During a data restore, the value of the LOGFILES configuration must always be greater than or equal to the total number of logical-log files. At some point during the restore, the number of logical-log files exceeded the value of LOGFILES.

Action: Increase the value of LOGFILES in ONCONFIG.

```
The number of configured CPU poll threads exceeds NUMCPUVPS.
```

Cause: The number of in-line poll threads that you specified in the ONCONFIG configuration file exceeds the number of CPU virtual processors.

Action: Reduce the number of in-line poll threads to be less than or equal to the number of CPU virtual processors.

onconfig parameter *parameter* modified from *old_value* to *new_value*

Cause: When the database server shared memory is reinitialized, this message documents any changes that occurred since the last initialization.

Action: None required.

oninit: Cannot have SINGLE_CPU_VP non-zero and number of CPU VPs greater than 1.

Cause: The ONCONFIG file contains VPCLASS cpu with a num= value greater than 1, and a nonzero value for SINGLE_CPU_VP. SINGLE_CPU_VP must be 0 (or omitted) when there are more than 1 CPU VPs.

Action: Correct the ONCONFIG file and restart the database server.

oninit: Cannot have SINGLE_CPU_VP non-zero and user-defined VP classes.

Cause: The ONCONFIG file contains a user-defined VPCLASS as well as a nonzero value for SINGLE_CPU_VP. SINGLE_CPU_VP must be 0 (or omitted) when the ONCONFIG file contains a user-defined VPCLASS.

Action: Correct the ONCONFIG file and restart the database server.

oninit: Cannot mix VPCLASS cpu and NUMCPUVPS, AFF_SPROC, AFF_NPROCS, or NOAGE parameters

Cause: The ONCONFIG file contains both VPCLASS cpu and one or more of the other listed parameters. It cannot contain both.

Action: Correct the ONCONFIG file and restart the database server.

```
oninit: Cannot mix VPCLASS aio and NUMAIOVPS parameters.
```

Cause: The ONCONFIG file contains both VPCLASS aio and NUMAIOVPS. It cannot contain both.

Action: Correct the ONCONFIG file and restart the database server.

```
oninit: Fatal error in initializing ASF with 'ASF_INIT_DATA'
flags asfcode = '25507'
```

Cause: The **nettype** specified in the **sqlhosts** file for the database server is invalid or unsupported, or the **servicename** specified in the **sqlhosts** file for the database server is invalid.

Action: Check the **nettype** and **servicename** values in the **sqlhosts** file for each DBSERVERNAME and for the DBSERVERALIASES. Check the **nettype** value in each NETTYPE parameter in the ONCONFIG file.

```
oninit: invalid or missing name for Subsystem Staging Blobspace.
```

Cause: You set the configuration parameter STAGEBLOB to a blobspace that does not exist.

Action: Use the **-d** option of **onspaces** to create the blobspace specified in STAGEBLOB, and restart the database server.

```
oninit: Too many VPCLASS parameters specified.
```

Cause: Too many VPCLASS parameter lines have been specified in the ONCONFIG file.

Action: Reduce the number of VPCLASS lines, if possible. If not possible, contact technical support.

oninit: VPCLASS classname bad affinity specification

Cause: The affinity specification for the VPCLASS line is incorrect. Affinity is specified as a range:

m use processor m.

m-n use processors in the range *m* to *n* inclusive, where *m* <= *n*, *m* >= 0, and *n* >= 0.

Action: Correct the VPCLASS parameter in the ONCONFIG file and restart the database server.

oninit: VPCLASS *classname* duplicate class name.

Cause: The VPCLASS *classname* in the ONCONFIG file has a duplicate name. VP class names must be unique.

Action: Correct the duplicate name and restart the database server.

oninit: VPCLASS *classname* illegal option

Cause: One of the fields in the VPCLASS *classname* parameter is illegal.

Action: Correct the parameter in the ONCONFIG file and restart the database server.

oninit: VPCLASS *classname* maximum number of VPs is out of the range 0-10000.

Cause: The maximum number of VPs specified by a VPCLASS parameter line must be in the range 1 to 10,000.

Action: Correct the value and restart the database server.

oninit: VPCLASS *classname* name is too long. Maximum length is *maxlength*.

Cause: The length of the name field in VPCLASS *classname* is too long.

Action: Choose a shorter class name, correct the ONCONFIG file, and restart the database server.

oninit: VPCLASS *classname* number of VPs is greater than the maximum specified

Cause: The initial number of VPs specified by a VPCLASS parameter is greater than the maximum specified by the same VPCLASS parameter.

Action: Correct the VPCLASS parameter and restart the database server.

oninit: VPCLASS *classname* number of VPs is out of the range 0-10000.

Cause: The initial number of VPs specified by a VPCLASS parameter line must be in the range 1 to 10,000.

Action: Correct the value and restart the database server.

onmode: VPCLASS *classname* name is too long. Maximum length is *maxlength.*

Cause: The name of a dynamically added VP class specified by **onmode -p** is too long.

Action: Pick a shorter name, and retry the **onmode -p** command.

Optical Subsystem is running.

Cause: You have set the value of the STAGEBLOB parameter in the configuration file, and the database server is communicating properly with the optical-storage subsystem.

Action: None required.

Optical Subsystem is not running.

Cause: You have set the value of the STAGEBLOB parameter in the configuration file, but the database server cannot detect the existence of the optical-storage subsystem.

Action: Check that the optical subsystem is on-line.

```
Optical Subsystem STARTUP Error.
```

Cause: The database server detects that the optical-storage subsystem is running, but the database server cannot communicate with it properly.

Action: Check your optical subsystem for errors.

```
On-Line Mode.
```

Cause: The database server is in on-line mode. Users can access all databases

Action: This is a status message. No action is required.

```
onspaces: unable to reset dataskip
```

Cause: This error message comes from the **onspaces** utility. For some reason, the utility cannot change the specification of DATASKIP (ON or OFF) across all dbspaces in the database server instance.

Action: You are unlikely to receive this message. If the error persists after you restart the database server, contact Informix Technical Support.

```
Open transaction detected when changing log versions.
```

Cause: The database server detected an open transaction while it was trying to convert the data from a previous version of the database server.

Action: Conversion is not allowed unless the last record in the log is a checkpoint. You must restore the previous version of the database server, force a checkpoint, and then retry conversion.

```
out of message shared memory
```

Cause: Could not allocate more memory for the specified segment.

Action: See the log file for additional information.

`out of resident shared memory`

Cause: Could not allocate more memory for the specified segment.

Action: See the log file for additional information.

`out of virtual shared memory`

Cause: Could not allocate more memory for the specified segment.

Action: See the log file for additional information.

`PANIC: Attempting to bring system down`

Cause: A fatal database server error occurred.

Action: See the error that caused the panic and attempt the corrective action suggested by the error message. Refer also to other messages in the message-log file for additional information that might explain the failure.

`Participant site` *`database_server`* `heuristically rolled back`

Cause: A remote site rolled back a transaction after it reached the prepared-for-commit phase.

Action: You might need to roll back the transaction on other sites and then restart it.

`Possible mixed transaction result.`

Cause: This message indicates that error -716 has been returned. Associated with this message is a list of the database servers where the result of a transaction is unknown.

Action: See "Determine If a Transaction Was Implemented Inconsistently" on page 31-4.

Prepared participant site *server_name* did not respond

Cause: Too many attempts were made to contact remote site *server_name.* After several timeout intervals were met, the site was determined to be down.

Action: Verify that the remote site is on-line and that it is correctly configured for distributed transactions. Once the remote site is ready, reinitiate the transaction.

Prepared participant site *server_name* not responding

Cause: The database server is attempting to contact remote site *server_name*. For some unknown reason, the database server cannot contact the remote site.

Action: Verify that the remote site is on-line and that it is correctly configured for distributed transactions.

Messages: Q-R-S

Quiescent Mode

Cause: The database server has entered quiescent mode from some other state. Only users logged in as **informix** or as **root** can interact with the database server. No user can access a database.

Action: None required.

Recovery Mode

Cause: The database server entered the recovery mode. No user can access a database until recovery is complete.

Action: None required.

Reversion cancelled

Cause: Errors encountered caused the reversion process to be cancelled.

Action: Correct the cause of the errors, and restart reversion.

```
Reversion complete
```

Cause: The reversion process was completed successfully.

Action: None required.

```
Recreating index: 'dbsname:"owner".tabname-idxname'
```

Cause: This message indicates which index is currently being re-created.

Action: None required.

```
Rollforward of log record failed, iserrno = nn
```

Cause: The message appears if, during fast recovery or a data restore, the database server cannot roll forward a specific logical-log record.The database server might be able to change to quiescent or on-line mode, but some inconsistency could result. See the message that immediately precedes this one for further information. The **iserrno** value is the RSAM error number.

Action: Contact Informix Technical Support.

```
scan_logundo: subsys ss, type tt, iserrno ee
```

Description: The variables in this message have the following values:

- *ss* is the logical-log record type.
- *tt* is the logical-log record type.
- *ee* is the iserror number.

Cause: A log undo failed because log type *tt* is corrupt.

Action: Examine the logical log using the **onlog** utility to determine if any action is needed. Contact Informix Technical Support.

`Session completed abnormally. Committing tx id 0xm, flags 0xn`

Cause: Abnormal session completion occurs only when the database server is attempting to commit a transaction that has no current owner, and the transaction develops into a long transaction.The database server forked a thread to complete the commit.

Action: None required.

`Session completed abnormally. Rolling back tx id 0xm, flags 0xn`

Cause: Abnormal session completion occurs only when the database server is attempting to commit a distributed transaction that has no current owner, and the transaction develops into a long transaction.The database server forked a thread that rolled back the transaction.

Action: None required.

`semctl: errno = nn`

Cause: While initializing a semaphore, an error occurred. The UNIX operating-system error is returned.

Action: See your operating-system documentation.

`semget: errno = nn`

Cause: An allocation of a semaphore set failed. The UNIX operating-system error is returned.

Action: See your operating-system documentation.

shmat: *some_string os_errno*: *os_err_text*

Cause: An attempt to attach to a shared-memory segment failed. The system error number and the suggested corrective action are returned.

Action: Review the corrective action (if given), and determine if it is reasonable to try. Also consult your operating-system documentation for more information.

shmctl: errno = *nn*

Cause: An error occurred while the database server tried to remove or lock a shared-memory segment. The UNIX operating-system error number is returned.

Action: See your operating-system documentation.

shmdt: errno = *nn*

Cause: An error occurred while the database server was trying to detach from a shared-memory segment. The UNIX operating-system error number is returned.

Action: See your operating-system documentation.

shmem sent to *filename*

Cause: The database server wrote a copy of shared memory to the specified file as a consequence of an assertion failure.

Action: None.

shmget: *some_str os_errno*: key *shmkey*: *some_string*

Cause: Either the creation of a shared-memory segment failed or an attempt to get the shmid associated with a certain key failed. The system error number and the suggested corrective action are returned.

Action: Consult your operating-system documentation.

`Shutdown Mode`

Cause: The database server is in the process of moving from on-line mode to quiescent mode.

Action: None required.

`Space` *`spacename`* `-- Recovery Begins(`*`addr`*`)`

Description: The variables in this message have the following values:

spacename	is the name of the dbspace or blobspace that the database server is recovering.
addr	is the address of the RSAM control block.

Cause: This informational message indicates that the database server is attempting to recover the space.

Action: None required.

`Space` *`spacename`* `-- Recovery Complete(`*`addr`*`)`

Description: The variables in this message have the following values:

spacename	is the name of the dbspace or blobspace that the database server has recovered.
addr	is the address of the RSAM control block.

Cause: This informational message indicates that the database server recovered the space.

Action: None required.

Space *spacename* -- Recovery Failed(*addr*)

Description: The variables in this message have the following values:

spacename	is the name of the dbspace or blobspace that the database server failed to recover.
addr	is the address of the RSAM control block.

Cause: This informational message indicates that the database server was unable to recover the space.

Action: None required.

sysmaster database built successfully

Cause: The database server successfully built the **sysmaster** database.

Action: None required.

Messages: T-U-V

Table *tablename* set to resident

Cause: The indicated table has been made resident by the SET TABLE statement.

Action: None required.

Table *tablename* and all its indexes set to resident

Cause: The indicated table and all its indexes have been made resident by the SET TABLE statement.

Action: None required.

```
The following tables have outstanding old version data
pages due to an In-Place Alter Table. Perform
UPDATE table-name SET column = column WHERE 1=1;
to clear these pages from the following tables.
```

Cause: Reversion to a previous version of the database server has been attempted while an in-place ALTER TABLE is in progress. The previous versions of the database server are unable to handle tables that have multiple schemas of rows in them.

Action: Force any in-place alters to complete by updating the rows in the affected tables before you attempt to revert to a previous version of the database server. To do this, create a dummy update in which a column in the table is set to its own value, forcing the row to be updated to the latest schema in the process without actually changing column values. Rows are always altered to the latest schema, so a single pass through the table that updates all rows will complete all outstanding in-place alters.

```
TIMER VP: Could not redirect I/O in initialization, errno = nn
```

Cause: Could not open /**dev/null** or duplicate the file descriptor associated with the opening of /**dev/null**. The system error number is returned.

Action: See your operating-system documentation.

```
Too Many Active Transactions.
```

Cause: During a data restore, there were too many active transactions. At some point during the restore, the number of active transactions exceeded 32 kilobytes.

Action: None.

```
Transaction Not Found
```

Cause: The logical log is corrupt. This situation can occur when a new transaction is started, but the first logical-log record for the transaction is not a BEGWORK record.

Action: Contact Informix Technical Support.

```
Transaction heuristically rolled back
```

Cause: A heuristic decision occurred to roll back a transaction after it completed the first phase of a two-phase commit.

Action: None required.

```
Transaction table overflow - user id nn, process id nn
```

Cause: A thread attempted to allocate an entry in the transaction table when no entries in the shared-memory table were available. The user ID and process ID of the requesting thread are displayed.

Action: Try again later.

```
Unable to create output file filename errno = nn
```

Cause: Cannot create output file *filename*. The *errno* value is the number of the UNIX error returned.

Action: Verify that the directory exists and has write permissions.

```
Unable to extend nn reserved pages for purpose in root chunk.
```

Cause: Cannot extend to *nn* reserved pages for *purpose* in root chunk. (The value *purpose* can be either Checkpoint/Log, DBSpace, Chunk, or Mirror Chunk).

Action: Reduce the ONCONFIG parameter for the resource cited; bring the database server up and free some space in the primary root chunk. Then reattempt the same operation.

```
Unable to initiate communications with the Optical Subsystem.
```

Cause: The optical driver supplied by the optical-drive vendor has indicated that drive is not accessible.

Action: Check driver installation and cabling between the computer and the drive.

`Unable to start SQL engine`

Cause: The database server encountered an out-of-memory condition.

Action: No action is necessary.

`Unable to open tblspace nn, iserrno = nn`

Cause: The database server cannot open the specified tblspace. (The value *nn* is the hexadecimal representation of the tblspace number.)

Action: See the ISAM error message number *nn*, which should explain why the tblspace cannot be accessed. The error message is contained in *Informix Error Messages* in Answers OnLine.

`Virtual processor limit exceeded`

Cause: You configured the database server with more than the maximum number of virtual processors allowed (1000).

Action: To reduce the number of virtual processors, decrease the values of VPCLASS, NUMCPUVPS, NUMAIOVPS, or NETTYPE in your ONCONFIG file.

`VPCLASS classname name is too long. Maximum length is maxlength.`

Cause: Indicates an internal error.

Action: Contact Informix Technical Support.

`VPCLASS classname duplicate class name.`

Cause: Indicates an internal error.

Action: Contact Informix Technical Support.

VPCLASS *classname* Not enough physical procs for affinity

Cause: The physical processors in the affinity specification for the VP class *classname* do not exist or are off-line. The problem might be with the VPCLASS parameter for cpu class VPs or with the AFF_SPROC and AFF_NPROCS parameters.

Action: Make sure the named processors are on-line. Correct the affinity specification for the named VP class. Restart the database server.

Messages: W-X-Y-Z

WARNING: aio_wait: errno = *nn*

Cause: While the database server was waiting for an I/O request to complete, it generated error number *nn* on an operation that it was attempting to execute.

Action: Contact Informix Technical Support for assistance.

WARNING: Buffer pool size may cause Dynamic Server to get into a locked state. Recommended minimum buffer pool size is *num* times maximum concurrent user threads.

Cause: There are not enough buffers in the buffer pool.The database server could use all available buffers and cause a deadlock to occur.

Action: Change the BUFFERS parameter in the ONCONFIG file to the number that this message recommends. For more information on the BUFFERS parameter, see "BUFFERS" on page 33-12.

warning: chunk timestamps are invalid.

Cause: A sanity check is performed on chunks when they are first opened at system initialization. The chunk specified did not pass the check and will be brought off-line.

Action: Restore the chunk from a dbspace backup or its mirror.

`Warning: unable to allocate requested big buffer of size nn`

Cause: The internal memory allocation for a big buffer failed.

Action: Increase either virtual memory size (SHMVIRTSIZE), the size of the added segments (SHMADD), or your total shared-memory size (SHMTOTAL).

`You must drop the database before reversion can complete.`

Cause: NLS or GLS enabled databases are not supported by Version 5.0.

Action: Drop the NLS or GLS databases, and start reversion.

`You must turn off data replication before reversion can complete.`

Cause: Data replication is not supported in Version 5.0. Also, the operations that occur during reversion do not work with data replication.

Action: Turn off data replication, and restart reversion.

Messages: Symbols

`... dropping sysmaster database`

Cause: Dropping **sysmaster** database during the reversion process.

Action: None.

`... reverting indexes`

Cause: Reverting indexes back to pre-6.0 format.

Action: None.

`... reverting reserved pages`

Cause: Reverting reserved pages.

Action: None.

```
... reverting tables that underwent In-Place Alter
```

Cause: Reverting tables that underwent in-place alter.

Action: None.

```
HH:MM:SS Informix Dynamic Server Version R.VV.PPPPP Software
Serial Number RDS#XYYYYYY
```

Cause: Startup of the database server, after the initialization of shared memory.

Action: None.

argument: invalid argument

Cause: This internal error indicates that an invalid argument was passed to an internal Informix routine.

Action: Contact Informix Technical Support.

function_name: cannot allocate memory

Cause: Could not allocate memory from internal shared-memory pool.

Action: Increase either virtual-memory size (SHMVIRTSIZE), the size of the added segments (SHMADD), or your total shared-memory size (SHMTOTAL).

Chapter

37

Interpreting Logical-Log Records

Reading Logical-Log Records 37-3
Transactions That Drop a Table or Index 37-4
Transactions That Are Rolled Back 37-4
Checkpoints with Active Transactions 37-5
Distributed Transactions 37-5

Logical-Log Record Structure 37-6
Logical-Log Record Header 37-6
Logical-Log Record Types and Additional Columns 37-7

The **onlog** utility and the ON-Archive command LIST/LOGRECORDS display the logical-log records that the logical-log files contain. For information about **onlog**, see "onlog: Display Logical-Log Contents" on page 35-23. For information about ON-Archive, see your *Archive and Backup Guide*.

This chapter provides the following information:

- Brief guidance on reading logical-log records
- A listing of the different logical-log record types

In general, you do not need to read and interpret your logical-log files. However, **onlog** output is useful in debugging situations. For example, you might want to use **onlog** to track a specific transaction or to see what changes the database server made to a specific tblspace. You can also use **onlog** to investigate the cause of an error that occurs during a rollforward. For more information about the specific instances in which you can use **onlog**, see Chapter 35, "Utilities."

Reading Logical-Log Records

Most SQL statements generate multiple logical-log records. Interpreting logical-log records is more complicated when the database server records the following events in the logical log:

- A transaction that drops a table or index
- A transaction that rolls back
- A checkpoint in which transactions are still active
- A distributed transaction

The logical-log records for these events are discussed briefly in the following sections.

Transactions That Drop a Table or Index

Once the database server drops a table or index from a database, it cannot roll back that drop operation. If a transaction contains a DROP TABLE or DROP INDEX statement, the database server handles this transaction as follows:

1. The database server completes all the other parts of the transaction and writes the relevant logical-log records.
2. The database server writes a BEGCOM record to the logical log and the records associated with the DROP TABLE or DROP INDEX (DINDEX, for example).
3. The database server writes a COMMIT record.

If the transaction is terminated unexpectedly after the database server writes the BEGCOM record to the logical log, the database server rolls *forward* this transaction during recovery because the drop operation cannot be rolled back.

Transactions That Are Rolled Back

When a rollback occurs, the database server generates a compensation-log record (CLR) for each record in the logical log that is rolled back. The database server uses the CLRs if a system failure takes place *during a rollback*. The CLRs provide the database server with information on how far the rollback progressed before the failure occurred. In other words, the database server uses the CLRs to log the rollback.

If the phrase `includes next record` is contained in a CLR, the next log record that is printed is included within the CLR log record as the compensating operation. Otherwise, you must assume that the compensating operation is the logical undo of the log record to which the **link** field of the CLR points.

Checkpoints with Active Transactions

If any transactions are active at the time of a checkpoint, checkpoint records include subentries that describe each of the active transactions using the following columns:

- Log begin (decimal format)
- Transaction ID (decimal format)
- Unique log number (decimal format)
- Log position (hexadecimal format)
- User name

Distributed Transactions

When distributed transactions (transactions that span multiple database servers) generate log records, they are slightly different than nondistributed transactions. You might need to read and interpret them to determine the state of the transaction on both database servers if a failure occurs as a transaction was committing.

The following log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

For more information about this type of logical-log record, see "Two-Phase Commit and Logical-Log Records" on page 30-30.

If you are performing distributed transactions with INFORMIX-TP/XA, the database server uses a XAPREPARE record instead of a PREPARE record.

Logical-Log Record Structure

Each logical-log record has *header* information. Depending on the record type, additional columns of information also appear in the output, as explained in "Logical-Log Record Types and Additional Columns" on page 37-7.

Logical-Log Record Header

Figure 37-1 contains sample output that illustrates the header columns that display for a logical-log record.

Figure 37-1
Sample Output from onlog

addr	len	type	xid	id	link
2c018	32	BEGIN	6	3	0
2c038	140	HDELETE	6	0	2c018
2c0c4	64	DELITEM	6	0	2c038
2c104	40	DELITEM	6	0	2c0c4
2c12c	72	HDELETE	6	0	2c104
2c174	44	DELITEM	6	0	2c12c
2c1a0	72	HDELETE	6	0	2c174
2c1e8	44	DELITEM	6	0	2c1a0
2c214	64	HDELETE	6	0	2c1e8
2c254	56	DELITEM	6	0	2c214
2c28c	48	DELITEM	6	0	2c254
2c2bc	24	PERASE	6	0	2c28c
2c2d4	20	BEGCOM	6	0	2c2bc

(1 of 2)

addr	len	type	xid	id	link
2c2e8	24	ERASE	6	0	2c2d4
2c300	28	CHFREE	6	0	2c2e8
2c31c	24	COMMIT	6	0	2c300

(2 of 2)

Figure 37-2 defines the contents of each header column.

Figure 37-2
Definition of onlog Header Columns

Header Field	Contents	Format
addr	Log-record address	Hexadecimal
len	Record length	Decimal
type	Record-type name	ASCII
xid	Transaction number	Decimal
id	Logical-log number	Decimal
link	Link to the previous record in the transaction	Hexadecimal

Logical-Log Record Types and Additional Columns

In addition to the six header columns that display for every record, some record types display additional columns of information. The information that is displayed varies, depending on record type. Figure 37-3 on page 37-8 lists all the record types and their additional columns.

The **Action** column indicates the type of database server action that generated the log entry. The **Additional Columns** and **Format** columns describe what is displayed for each record type in addition to the header described in "Logical-Log Record Header" on page 37-6.

Figure 37-3
Logical-Log Record Types

Record Type	Action	Additional Columns	Format
ADDCHK	Add chunk.	chunk number	decimal
		chunk name	ASCII
ADDDBS	Add dbspace.	dbspace name	ASCII
ADDITEM	Add item to index.	tblspace ID	hexadecimal
		rowid	hexadecimal
		logical page	decimal
		key number	decimal
		key length	decimal
ADDLOG	Add log.	log number	decimal
		log size (pages)	decimal
		pageno	hexadecimal
ALTERDONE	Alter of fragment complete	tblspace ID	hexadecimal
		physical page number previous page	hexadecimal
		logical page number	decimal
		version of alter	decimal
BADIDX	Bad index.	tblspace ID	hexadecimal
BEGCOM	Begin commit.	(none)	(none)
BEGIN	Begin work.	date	decimal
		time	decimal
		PID	decimal
		user	ASCII
BEGPREP	Written by the coordinator database server to record the start of the two-phase commit protocol.	flags	decimal (value is 0 in a distributed transaction)
		no. of participants	decimal

(1 of 13)

Record Type	Action	Additional Columns	Format
BFRMAP	Blob free-map change.	tblspace ID	hexadecimal
		bpageno	hexadecimal
		status	USED / FREE
		log ID	decimal
		prev page	hexadecimal
BLDCL	Build tblspace.	tblspace ID	hexadecimal
		fextsize	decimal
		nextsize	decimal
		row size	decimal
		ncolumns	decimal
		table name	ASCII
BMAPFULL	Bitmap modified to prepare for alter.	tblspace ID	hexadecimal
		bitmap page num	decimal
BMAP2TO4	2-bit bitmap altered to two 4-bit bitmaps.	tblspace ID	hexadecimal
		2-bit bitmap page number	decimal
		flags	decimal
BSPADD	Add blobspace.	blobspace name	ASCII
BTCPYBCK	Copy back child key to parent.	tblspace ID	hexadecimal
		parent logical page	decimal
		child logical page	decimal
		slot	decimal
		rowoff	decimal
		key number	decimal

(2 of 13)

Record Type	Action	Additional Columns	Format
BTMERGE	Merge B-tree nodes.	tblspace ID	hexadecimal
		parent logical page	decimal
		left logical page	decimal
		right logical page	decimal
		left slot	decimal
		left rowoff	decimal
		right slot	decimal
		right rowoff	decimal
		key number	decimal
BTSHUFFL	Shuffle B-tree nodes.	tblspace ID	hexadecimal
		parent logical page	decimal
		left logical page	decimal
		right logical page	decimal
		left slot	decimal
		left rowoff	decimal
		key number	decimal
		flags	hexadecimal
BTSPLIT	Split B-tree node.	tblspace ID	hexadecimal
		rowid	hexadecimal
		parent logical page	decimal
		left logical page	decimal
		right logical page	decimal
		infinity logical page	decimal
		rootleft logical page	decimal
		midsplit	decimal
		key number	decimal
		key length	decimal
CHALLOC	Chunk extent allocation.	pageno	hexadecimal
		size	hexadecimal
CHCOMBINE	Chunk extent combine.	pageno	hexadecimal

(3 of 13)

Record Type	Action	Additional Columns	Format
CHFREE	Chunk extent free.	pageno	hexadecimal
		size	hexadecimal
CHPHYLOG	Change physical-log location.	pageno	hexadecimal
		size in kilobytes	hexadecimal
		dbspace name	ASCII
CHSPLIT	Chunk extent split.	pageno	hexadecimal
CINDEX	Create index.	tblspace ID	hexadecimal
		low rowid	decimal
		high rowid	decimal
		index descriptor	
CKPOINT	Checkpoint.	max users	decimal
		number of active transactions	decimal
CLR	Compensation-log record; part of a rollback.	(none)	(none)
CLUSIDX	Create clustered index.	tblspace ID	hexadecimal
		key number	decimal
COLREPAI	Adjustment of BYTE, TEXT, or VARCHAR column.	tblspace ID	hexadecimal
		no. of columns adjusted	decimal
COMMIT	Commit work.	date	decimal
		time	decimal
CREATE	Creation of smart large object.	smart large object ID,	hexadecimal
		number of extents in lomaphdr	decimal
DELETE	Delete before-image.	tblspace ID	hexadecimal
		rowid	hexadecimal

(4 of 13)

Record Type	Action	Additional Columns	Format
DELITEM	Delete item from index.	tblspace ID	hexadecimal
		rowid	hexadecimal
		logical page	decimal
		key number	decimal
		key length	decimal
DERASE	Drop partition in down dbspace.	tblspace ID	hexadecimal
		lockid	hexadecimal
DINDEX	Drop index.	tblspace ID	hexadecimal
		key number	decimal
DRPBSP	Drop blobspace.	blobspace name	ASCII
DRPCHK	Drop chunk.	chunk number	decimal
		chunk name	ASCII
DRPDBS	Drop dbspace.	dbspace name	ASCII
DRPLOG	Drop log.	log number	decimal
		log size (pages)	decimal
		pageno	hexadecimal

(5 of 13)

Record Type	Action	Additional Columns	Format
ENDTRANS	Written by both the coordinator and participant database servers to record the end of the transaction. ENDTRANS instructs the database server to remove the transaction entry from its shared-memory transaction table and close the transaction. In the coordinator logical log, each BEGPREP that results in a committed transaction is paired with an ENDTRANS record. If the final decision of the coordinator is to roll back the transaction, no ENDTRANS record is written. In the participant logical log, each ENDTRANS record is paired with a corresponding HEURTX record.	(none)	(none)
ERASE	Drop tblspace.	tblspace ID	hexadecimal
HDELETE	Home row delete.	tblspace ID rowid slotlen	hexadecimal hexadecimal decimal
HEURTX	Written by a participant database server to record a heuristic decision to roll back the transaction. It should be associated with a standard ROLLBACK record indicating that the transaction was rolled back.	flag	hexadecimal (value is always 1)

(6 of 13)

Record Type	Action	Additional Columns	Format
HINSERT	Home row insert.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
HUPAFT	Home row update, after-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
HUPBEF	Home row update, before-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
HUPDATE	If the home row update before-images and after-images can both fit into a single page, the database server writes a single HUPDATE record.	tblspace ID	hexadecimal
		rowid	hexadecimal
		forward ptr rowid	hexadecimal
		old slotlen	decimal
		new slotlen	decimal
		no. of pieces	decimal
IDXFLAGS	Index flags.	tblspace ID	hexadecimal
		key number	hexadecimal
INSERT	Insert after-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
LCKLVL	Locking mode.	tblspace ID	hexadecimal
		old lockmode	hexadecimal
		new lockmode	hexadecimal
LG_CDINDEX	Create detached index.	database name	ASCII
		owner	ASCII
		table name	ASCII
		index name	ASCII
LG_DERASE	Drop partition in disabled dbspace.	partition number	hexadecimal
		table lock number	decimal

(7 of 13)

Record Type	Action	Additional Columns	Format
MVIDXND	Index node moved to allow for 2-bit to 4-bit bitmap conversion.	tblspace ID	hexadecimal
		old page number	decimal
		new page number	decimal
		parent page number	decimal
		parent slot number	decimal
		parent slot offset	decimal
		key number	decimal
PBDELETE	Tblspace blobpage delete.	bpageno	hexadecimal
		status	USED / FREE
		unique ID	decimal
PBINSERT	Tblspace blobpage insert.	bpageno	hexadecimal
		tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
		pbrowid	hexadecimal
PDINDEX	Predrop index.	tblspace ID	hexadecimal
PGALTER	Page altered in-place.	tblspace ID	hexadecimal
		physical page number	hexadecimal

(8 of 13)

Record Type	Action	Additional Columns	Format
PGMODE	Page mode modified in bitmap.	tblspace ID	hexadecimal
		logical page number	decimal
		old mode	hexadecimal
		new mode	hexadecimal
PERASE	Preerase old file.	tblspace ID	hexadecimal
PNLOCKID	Change partitions lockid.	tblspace ID	hexadecimal
		old lock ID	hexadecimal
		new lock ID	hexadecimal
PNSIZES	Set tblspace extent sizes.	tblspace ID	hexadecimal
		fextsize	decimal
		nextsize	decimal
PREPARE	Written by a participant database server to record the ability of the participant to commit the transaction, if so instructed.	DBSERVERNAME of coordinator	ASCII
PTADESC	Add of alter description information.	tblspace ID	hexadecimal
		physical page number of previous page	hexadecimal
		logical page number	decimal
		number of columns added	decimal

Record Type	Action	Additional Columns	Format
PTALTER	Alter of fragment begun.	tblspace ID	hexadecimal
		physical page number previous page	hexadecimal
		logical page number	decimal
		alter desc page number	decimal
		num columns added	decimal
		version of alter	decimal
		added rowsize	decimal
PTCOLUMN	Add of special columns to fragment.	tblspace ID	hexadecimal
		number of columns	decimal
PTEXTEND	Partition extend.	tblspace ID	hexadecimal
		last logical page	decimal
		first physical page	hexadecimal
PTRENAME	Rename table.	tblspace ID	hexadecimal
		old table name	ASCII
		new table name	ASCII
RDELETE	Remainder page delete.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
REVERT	Logs the reversion of a database space to a database space of an earlier version.	type of reversion event	decimal
		arg1	decimal
		arg2	decimal
		arg3	decimal
RINSERT	Remainder page insert.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal

(10 of 13)

Record Type	Action	Additional Columns	Format
ROLLBACK	Rollback work.	date time	decimal decimal
RSVEXTEN	Logs the extension to the reserved pages.	no. of pages physical page number of extent	decimal hexadecimal
RUPAFT	Remainder page update, after-image.	tblspace ID rowid slotlen	hexadecimal hexadecimal decimal
RUPBEF	Remainder page update, before-image.	tblspace ID rowid slotlen	hexadecimal hexadecimal decimal
RUPDATE	If the remainder page update before-images and after-images can both fit into a single page, the database server writes a single RUPDATE record.	tblspace ID rowid forward ptr rowid old slotlen new slotlen no. of pieces	hexadecimal hexadecimal hexadecimal decimal decimal decimal
SYNC	Written to a logical-log file if that log file is empty and administrator instructs the database server to switch to next log file.	(none)	(none)

(11 of 13)

Record Type	Action	Additional Columns	Format
TABLOCKS	Written by either a coordinator or a participant database server. It is associated with either a BEGPREP or a PREPARE record and contains a list of the locked tblspaces (by tblspace number) held by the transaction. (In a distributed transaction, transactions are shown as the owners of locks.)	no. of locks tblspace number	decimal hexadecimal
UDINSERT	Append of new user data	chunk page within chunk offset within page data length	decimal hexadecimal hexadecimal hexadecimal
UDUPAFT	Update of user data after image if a UDWRITE is too expensive.	chunk page within chunk offset within page data length	decimal hexadecimal hexadecimal hexadecimal
UDUPBEF	Update of user data before image if a UDWRITE is too expensive.	chunk page within chunk offset within page data length	decimal hexadecimal hexadecimal hexadecimal
UDWRITE	Update of user data (difference image).	chunk page within chunk offset within chunk length before write length after write	decimal hexadecimal hexadecimal hexadecimal hexadecimal
UNDO	Header record to a series of transactions to be rolled back.	count	decimal

Record Type	Action	Additional Columns	Format
UNDOBLDC	This record is written if a CREATE TABLE statement should be rolled back but cannot be because the relevant chunk is down. When the log file is replayed, the table will be dropped.	tblspace number	hexadecimal
UNIQID	Logged when a new serial value is assigned to a row.	tblspace ID unique ID	hexadecimal decimal
UPDAFT	Update after-image.	tblspace ID rowid	hexadecimal hexadecimal
UPDBEF	Update before-image.	tblspace ID rowid	hexadecimal hexadecimal
XAPREPARE	Participant can commit this XA transaction.	(none)	(none)

(13 of 13)

Chapter

38

Disk Structures and Storage

Dbspace Structure and Storage 38-4
Structure of the Root Dbspace 38-4
Reserved Pages 38-6
Structure of a Regular Dbspace 38-15
Structure of an Additional Dbspace Chunk 38-16
Structure of a Mirrored Chunk 38-17
Structure of the Chunk Free-List Page 38-18
Chunk Free-List Page Entries 38-18
Creation of Free-List Pages 38-19
Structure of the Tblspace Tblspace 38-19
Tblspace Tblspace Entries 38-20
What Is the Tblspace Number? 38-21
Tblspace Number Elements 38-22
Tblspace Tblspace Size 38-23
Tblspace Tblspace Bit-Map Page 38-23
Structure of the Database Tblspace 38-24
What Is the Database Tblspace Number? 38-24
Database Tblspace Entries 38-24
Structure of a Dbspace Bit-Map Page 38-25
Types of Bit-Map Entries 38-25
Two-Bit Bit-Map Pages 38-26
Four-Bit Bit-Map Pages 38-26
Structure and Allocation of an Extent 38-27
Extent Structure 38-28
Next-Extent Allocation 38-31
Structure and Storage of a Dbspace Page 38-34
Structure of a Dbspace Page 38-34
Rowid in Nonfragmented Tables 38-37
Rowid in Fragmented Tables 38-39
Informix Recommendations on Use of Rowid 38-40
Data-Row Format and Storage 38-40

Structure of B-Tree Index Pages 38-48
Definition of Terms in B-Tree Indexing 38-48
Logical Storage of Indexes 38-50
Physical Storage Format of Index Pages 38-55

Blobspace Structure and Storage 38-59
Structure of a Blobspace 38-59
TEXT and BYTE Data Storage and the Descriptor 38-61
When Are TEXT or BYTE Objects Created? 38-61
Are TEXT and BYTE Objects Modified? 38-62
What Limits the Size of TEXT and BYTE Objects? 38-62
Structure of a Dbspace Blobpage 38-62
Blobspace Page Types. 38-64
What Is the Blobspace Free-Map Page? 38-64
What Is the Blobspace Bit-Map Page? 38-64
What Is the Blobpage? 38-65
Structure of a Blobspace Blobpage 38-65
Blobpage Structure 38-65
What Is in the Blobpage Header? 38-66

Database and Table Creation: What Happens on Disk 38-68
Creating a Database 38-68
Disk-Space Allocation for System Catalog Tables 38-68
System Catalog Tables Are Tracked. 38-69
Creating a Table. 38-69
Disk-Space Allocation 38-70
Entry Is Added to Tblspace Tblspace 38-70
Entries Are Added to the System Catalog Tables 38-70
What Happens on Disk When a Temporary Table Is Created? 38-72

The database server achieves its high performance by managing its own I/O. The database server manages storage, search, and retrieval. As the database server stores data, it creates the structures it needs to search for and retrieve the data later. The database server disk structures also store and track control information needed to manage logging and backups. Database-server structures contain all the information needed to ensure data consistency, both physical and logical.

Before you read this chapter, familiarize yourself with the disk-space terms and definitions in Chapter 13, "Where Is Data Stored?"

This chapter discusses the following topics related to disk data structures:

- TEXT and BYTE storage and descriptors
- Blobspace blobpages
- Blobspace page types
- Blobspace structure
- Chunk free-list page structure
- Data row storage
- Database tblspace structure
- Dbspace bit-map page structure
- Dbspace blobpage structure
- Dbspace chunk structure
- Dbspace page structure
- Extent structure
- Index page structure

- Mirrored chunk structure
- Next-extent allocation
- Reserved pages
- Root dbspace structure
- Rowid structure
- Tblspace tblspace structure
- What happens when you create a database
- What happens when you create a table
- What happens when you create a temporary table

Dbspace Structure and Storage

This section explores the disk structures and storage techniques that the database server uses to store data in a dbspace.

Structure of the Root Dbspace

The ROOTNAME, ROOTOFFSET, ROOTPATH, and ROOTSIZE configuration parameters specify the size and location of the initial chunk of the root dbspace. If the root dbspace is mirrored, the MIRROROFFSET and MIRRORPATH configuration parameters specify the mirror-chunk location. For more information about these parameters, see Chapter 33, "Configuration Parameters."

As part of disk-space initialization, the database server initializes the following structures in the initial chunk of the root dbspace:

- Twelve reserved pages
- The first chunk free-list page
- The tblspace tblspace
- The database tblspace
- The physical log
- The logical-log files

Figure 38-1 illustrates the structures that reside in the root dbspace following disk-space initialization. Each of these structures is described in the paragraphs that follow. To see that your root dbspace follows this organization, execute the command **oncheck -pe**, which produces a dbspace usage report, by chunk.

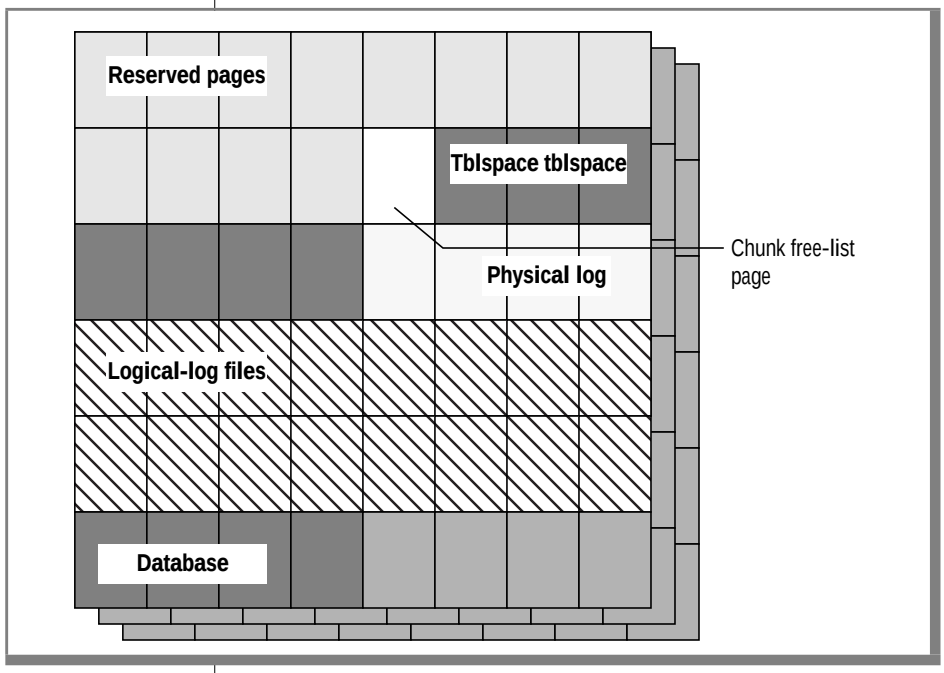


Figure 38-1
Initial Chunk in the Root Dbspace

Reserved Pages

The first 12 pages of the initial chunk of the root dbspace are reserved pages. Each reserved page contains specific control and tracking information used by the database server**.** Figure 38-2 provides the name and a brief description of each of the reserved pages.

Figure 38-2
Reserved Pages and Their Use

Order	Page Name	Page Usage
1	PAGE_PZERO	System identification
2	PAGE_CONFIG	Copy of the ONCONFIG file
3	PAGE_1CKPT	Checkpoint / logical-log tracking
4	PAGE_2CKPT	Alternate CKPT page
5	PAGE_1DBSP	Dbspace descriptions
6	PAGE_2DBSP	Alternate DBSP page
7	PAGE_1PCHUNK	Primary chunk descriptions
8	PAGE_2PCHUNK	Alternate PCHUNK page
9	PAGE_1MCHUNK	Mirrored-chunk descriptions
10	PAGE_2MCHUNK	Alternate MCHUNK page
11	PAGE_1ARCH	Dbspace backup tracking
12	PAGE_2ARCH	Alternate ARCH page

Reserved-Page Pairs

Beginning with the third reserved page, PAGE_1CKPT, the pages are organized into pairs. These pairs come into play when the database server begins to update the reserved pages as part of the checkpoint procedure.

The database server writes to PAGE_PZERO on the following three occasions:

- At boot time
- When you convert to Version 7.3 of the database server from Version 7.2x
- When you convert from Version 7.2x to a previous version of the database server

The reserved-page checkpoint information is stored in a two-page pair, PAGE_1CKPT and PAGE_2CKPT. This information changes for each checkpoint. During each checkpoint, the database server writes the latest checkpoint information to one of the pages in the pair. During the next checkpoint, the database server writes the information to the other page in the pair. At any point, one page in the checkpoint reserved-page pair contains a copy of information written at the most-recent checkpoint, and the other page in the pair contains a copy of information written at the second-most-recent checkpoint.

The database server follows a different procedure for updating information in the next three reserved-page pairs. The database server updates the dbspace, primary chunk, or mirror-chunk reserved pages only when a change occurs. The database server learns of a change from flags that are set on the dbspace, primary chunk, and mirror descriptor tables in shared memory. During the checkpoint, the database server checks each shared-memory descriptor table for a change flag.

If the flag is set, the database server prepares to write the modified descriptor information to the appropriate page in the reserved-page pair. First, the database server switches from the current page (which is the page that received the last write) to the other page in the pair. Second, the database server writes the information to the reserved page. Third, the database server updates the fields that contain the numbers of the current pages for the dbspace, primary chunk, or mirror-chunk information. These fields are located on the PAGE_1CKPT and PAGE_2CKPT pages.

The last pair of reserved pages contains dbspace backup and data-replication information. The database server updates PAGE_ARCH after a dbspace backup or a change in data-replication status.

To obtain a listing of the contents of your reserved pages, execute the command **oncheck -pr**.

PAGE_PZERO

The first reserved page in the root dbspace is PAGE_PZERO. The following table lists the PAGE_PZERO fields and definitions.

Field Name	Description
Identity	Database server copyright
Database system state	Unused
Database system flags	Unused
Page size	Page size for this computer, in bytes
Date / Time created	Date and time of disk-space initialization
Version number of creator	Reserved for internal use
Last modified time stamp	Unused

PAGE_CONFIG

The second reserved page in the root dbspace is PAGE_CONFIG. This page contains a copy of the ONCONFIG file. For information about this file, see Chapter 33, “Configuration Parameters.”

PAGE_CKPT

The third reserved page in the root dbspace is PAGE_1CKPT. The fourth reserved page, PAGE_2CKPT, is the second page in the pair.

The database server uses the checkpoint and logical-log file information for checkpoint processing. The date and time of the last checkpoint, available, is obtained from this reserved page.

UNIX

On UNIX, you can also get the date and time of the last checkpoint from the Force-Ckpt menu of ON-Monitor. ♦

The following table lists the checkpoint and logical-log file tracking fields and definitions.

Field Name	Description
Time stamp of checkpoint	Time stamp of the last checkpoint, displayed as decimal value
Time of checkpoint	Time that the last checkpoint occurred
Physical-log begin address	Beginning address of the physical log
Physical-log size	Number of pages in the physical log
Physical-log position at Ckpt	Physical location for the start of the next set of before-images
Logical-log unique identifier	ID number of the logical-log file that stores the most-recent checkpoint record
Logical-log position at Ckpt	Physical location of this checkpoint record in the logical-log file
Dbspace descriptor page	Address of the current dbspace reserved page
Chunk descriptor page	Address of the current primary-chunk reserved page
Mirrored-chunk descriptor page	Address of the current mirrored-chunk reserved page

The following table lists the fields that display for each logical-log file.

Field Name	Description
Log file number	Number of this logical-log file
Log file flags	Logical-log file flags (see next five entries)
0x01	Log file in use
0x02	Log file is the current log.
0x04	Log file has been backed up.
0x08	Log file is newly added.
0x10	Log file has been written to dbspace backup tape.
0x100	Blobspace is the optical staging blobspace.
0x200	Dbspace / blobspace is being physically recovered.
0x400	Is physically recovered
0x800	Is being logically recovered
0x1000	Dbspace has had a tblspace dropped since last checkpoint.
0x2000	Dbspace has had a tblspace dropped since last checkpoint.
0x4000	Dbspace is being backed up.
Time stamp	Time stamp when log filled (decimal)
Date / time file filled	Date and time that this log filled
Unique identifier	ID number of this logical-log file
Physical location	Address of this logical-log file on disk
Log size	Number of pages in this logical-log file
Number pages used	Number of pages used in this logical-log file

PAGE_DBSP

The fifth reserved page in the root dbspace is PAGE_1DBSP. The sixth reserved page, PAGE_2DBSP, is the second page in the pair.

The database server uses the dbspace page to describe each dbspace and its current status.

The following table lists the dbspace description fields and definitions.

Field Name	Description
Dbspace number	Dbspace number
Flags	Dbspace flags (see next six entries)
0x01	Dbspace is not mirrored.
0x02	Dbspace includes mirrored chunks.
0x04	Dbspace contains a down chunk.
0x08	Dbspace is newly mirrored.
0x10	Dbspace is a blobspace.
0x80	Blobspace has been dropped.
First chunk	Number of the dbspace initial chunk
Number of chunks	Number of chunks in the dbspace
Date / time created	Date and time that the dbspace was created
Dbspace name	Dbspace name
Dbspace owner	Dbspace owner

PAGE_PCHUNK

The seventh reserved page in the root dbspace is PAGE_1PCHUNK. The eighth reserved page, PAGE_2PCHUNK, is the second page in the pair.

The database server uses the primary chunk page to describe each chunk, its pathname, its relation to the dbspace, and its current status.

The following table lists the primary chunk fields and definitions.

Field Name	Description
Chunk number	Primary-chunk number
Next chunk in DBSpace	Number of the next chunk in the dbspace
Chunk offset	Offset of chunk, in pages
Chunk size	Number of pages in the chunk
Number of free pages	Number of free pages in the chunk
Dbspace number	Number of the dbspace for this chunk
Overhead	Free-map page address (blobspace only)
Flags	Chunk flags (see next 10 entries in this table)
0x01	Raw device
0x02	Block device
0x04	Operating-system file
0x08	Needs sync() to operating system
0x20	Chunk is off-line.
0x40	Chunk is on-line.
0x80	Chunk is in recovery.
0x100	Chunk is newly mirrored.
0x200	Chunk is part of a blobspace.
0x400	Chunk is being dropped.
0x800	Chunk is part of staging blobspace.
0x1000	Chunk is inconsistent.
Chunk name length	Length of the chunk pathname
Chunk path	Operating-system path for chunk

PAGE_MCHUNK

The ninth reserved page in the root dbspace is PAGE_1MCHUNK. The tenth reserved page, PAGE_2MCHUNK, is the second page in the pair.

The database server uses the mirrored-chunk page to describe each mirrored chunk, its pathname, its relation to the dbspace, and its current status.

The following table lists the mirror-chunk fields and definitions.

Field Name	Description
Primary-chunk number	Chunk number
Next chunk in DBSpace	Number of the next chunk in the dbspace
Chunk offset	Offset of chunk, in pages
Chunk size	Number of pages in the chunk
Number of free pages	Number of free pages in the chunk
DBSpace number	Number of the dbspace for this chunk
Overhead	Free-map page address (blobspace only)
Flags	Chunk flags (see next 10 entries in this table)
0x01	Raw device
0x02	Block device
0x04	Operating-system file
0x08	Needs sync() to operating system
0x10	Chunk is a mirrored chunk.
0x20	Chunk is off-line.
0x40	Chunk is on-line
0x80	Chunk is in recovery.
0x100	Chunk is newly mirrored.
0x200	Chunk is part of a blobspace.

(1 of 2)

Field Name	Description
0x400	Chunk is being dropped.
0x800	Chunk is part of staging blobspace.
0x1000	Chunk is inconsistent.
Chunk name length	Length of the chunk pathname
Chunk path	Operating-system path for chunk

(2 of 2)

PAGE_ARCH

The eleventh reserved page in the root dbspace is PAGE_1ARCH. The twelfth reserved page, PAGE_2ARCH, is the second page in the pair.

The database server uses the PAGE_ARCH reserved pages to describe the most-recent and the second-most-recent dbspace backups. The database server also uses PAGE_ARCH to record data about data replication.

The following table lists the dbspace backup and data-replication fields and definitions.

Field Name	Description
Archive level	Level of this dbspace backup (0, 1, or 2)
Real Time Archive Began	Date and time of this dbspace backup
Time Stamp Archive Began	Time stamp for this dbspace backup (decimal)
Logical Log Unique Id	ID number of the logical-log file that contains the record of this dbspace backup
Logical Log Position	Physical location of this checkpoint record in the logical-log file
DR Ckpt log id	ID number of the logical-log file that contains the most-recent common checkpoint

(1 of 2)

Field Name	Description
DR Ckpt Logical Pos	Logical-log file position of the most-recent common checkpoint
DR Last Logical Log Id	Logical-log file ID of the last log record sent or received
DR Last Logical Log Page	Logical-log file position of last log record sent/received

(2 of 2)

The database server only displays fields concerning data replication if data replication has been initialized.

Structure of a Regular Dbspace

After disk-space initialization, you can add new dbspaces. When you create a dbspace, you assign at least one chunk (either raw or cooked disk space) to the dbspace. This chunk is referred to as the initial chunk of the dbspace. Figure 38-3 illustrates the structure of the initial chunk of a regular (nonroot) dbspace.

When the dbspace is first created, it contains the following structures:

- The first chunk free-list page in the chunk
- The tblspace tblspace for this dbspace
- Unused pages

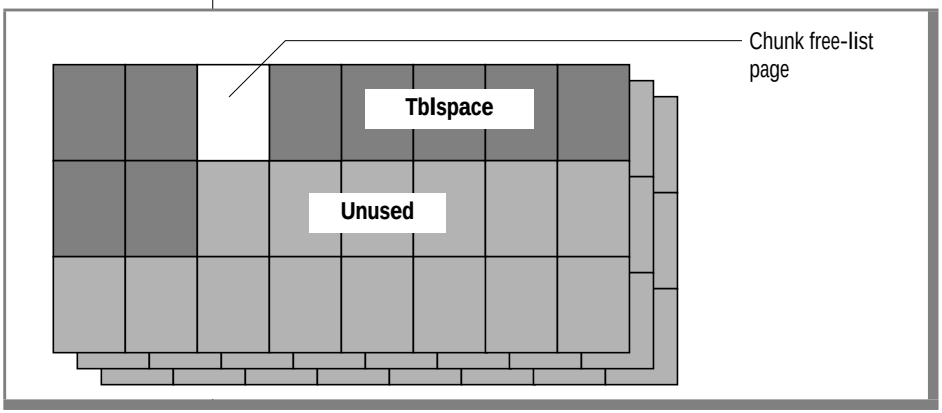


Figure 38-3
Initial Chunk of Regular Dbspace

Structure of an Additional Dbspace Chunk

You can create a dbspace that contains more than one chunk. The initial chunk in a dbspace contains the tblspace tblspace for the dbspace. Additional chunks do not. When an additional chunk is first created, it contains the following structures:

- Two reserved pages (reserved for future use)
- The first chunk free-list page
- Unused pages

Figure 38-4 illustrates the structure of all additional chunks in a dbspace. (The structure also applies to additional chunks in the root dbspace.)

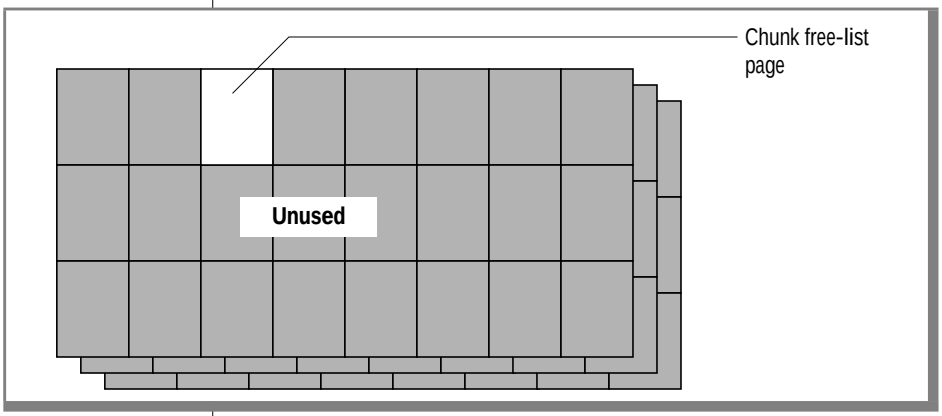


Figure 38-4
Additional Dbspace Chunk

Structure of a Mirrored Chunk

Each mirrored chunk must be the same size as its primary chunk. When a mirrored chunk is created, the database server schedules a thread to write the contents of the primary chunk to the mirrored chunk immediately.

The mirrored chunk contains the same control structures as the primary chunk. Mirrors of either blobspace or dbspace chunks contain the same physical contents as their primary counterpart after the database server brings them on-line.

Figure 38-5 illustrates the mirror-chunk structure as it appears after the chunk is created.

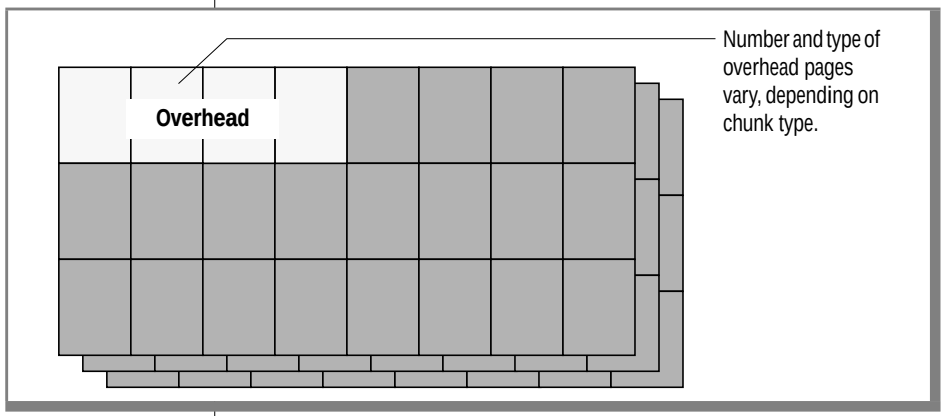


Figure 38-5
Mirror-Chunk Structure

The mirror-chunk structure is marked full as soon as it is created. For more information on this topic, see "What Is the Structure of a Mirrored Chunk?" on page 23-11.

Structure of the Chunk Free-List Page

In every chunk, the page that follows the last reserved page is the first of one or more chunk free-list pages that tracks available space in the chunk. A chunk free-list page contains the starting page (page offset into the chunk) of each section of free space and the length of the free space measured in number of pages. Figure 38-6 illustrates the location of the free-list page.

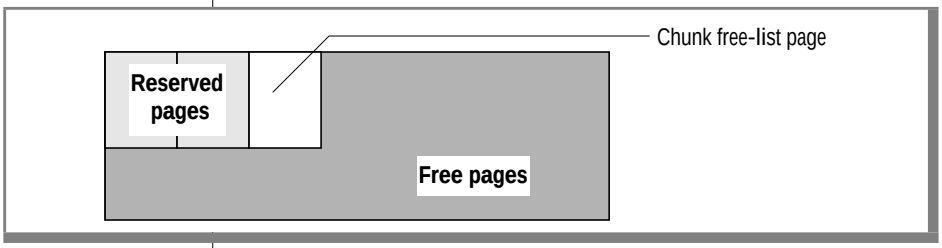


Figure 38-6
Free-List Page

Chunk Free-List Page Entries

Initially, the chunk free-list page has a single entry. For example, in any dbspace initial chunk other than root, the starting page number of free space is three. The first two pages are currently unused but reserved for future use by Informix. The third page is filled by the chunk free list. The length of the free space in the first entry is the size of the chunk minus three pages.

When chunk pages are allocated, the loss of free space is recorded by changing the starting-page offset and the length of the unused space.

When chunk pages are freed (for example, if a table is dropped), entries are added that describe the starting page and length of each section of newly freed, contiguous space.

If newly freed space is contiguous with existing free space, only the length of the existing entry is changed. Otherwise, a new entry is created.

Figure 38-7 illustrates a sample listing from a chunk free-list page.

Chunk Offset	Number of Free Pages
14	28
123	36
208	52

Figure 38-7
Sample Listing from a Chunk Free-List Page

Creation of Free-List Pages

If an additional chunk free-list page is needed to accommodate new entries, a new chunk free-list page is created in one of the free pages in the chunk. The chunk free-list pages are chained in a linked list. Each free-list page contains entries that describe all free space starting with the next page and continuing to the next chunk free-list page or to the end of the chunk.

Structure of the TblspaceTblspace

In the initial chunk of every dbspace, the page that follows the chunk free-list page is the first page of the tblspace tblspace. The tblspace tblspace is a collection of pages that describe the location and structure of all tblspaces in this dbspace. Figure 38-8 illustrates the location of the tblspace tblspace.

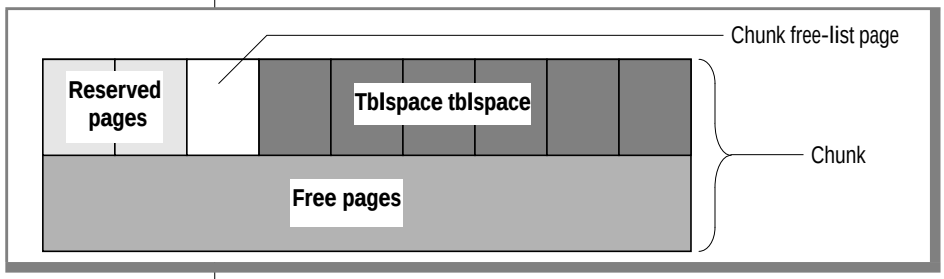


Figure 38-8
TblspaceTblspace

Tblspace Tblspace Entries

Each page in the tblspace tblspace describes one tblspace in the dbspace and is considered one entry. Entries in the tblspace tblspace are added when a new table is created or when a fragment is added to a fragmented table.

The first page in every tblspace tblspace is a bit map of the pages in the tblspace tblspace. The second page is the first tblspace entry, and it describes itself. The third page describes the first user-created table in this dbspace. Each tblspace tblspace entry (page) includes the following components.

Component	Description
Page header	24 bytes, standard page-header information
Page-ending time stamp	4 bytes
Tblspace header	56 bytes, general tblspace information available from an **oncheck -pt** display
Column information	Each special column in the table is tracked with an 8-byte entry. (A special column is defined as a VARCHAR, BYTE, or TEXT data type.)
Tblspace name	80 bytes, *database.owner.tablename*
Index information	Each index on the table is tracked with a 16-byte entry.
Index column information	Each column component in each index key is tracked with a 4-byte entry.
Extent information	Each extent allocated to this tblspace is tracked with an 8-byte entry.

What Is the TblSpace Number?

Each tblspace that is described in the tblspace tblspace receives a tblspace number. This tblspace number is the same value that is stored as the **partnum** field in the **systables** system catalog table and as the **partn** field in the **sysfragments** system catalog table.

The tblspace number (**partnum**) is stored as an integer (4 bytes). The following SQL query retrieves the **partnum** for every table in the database (these can be located in several different dbspaces) and displays it with the table name and the hexadecimal representation of **partnum**:

```
SELECT tabname, tabid, partnum, HEX(partnum) hex_tblspace_name FROM systables
```

If the output includes a row with a table name but a **partnum** of 0, this table consists of two or more table fragments, each located in its own tblspace. For example, Figure 38-9 shows a table called **account** that has **partnum** 0.

```
tabname              tabid     partnum     hex_tblspace_name

sysfragments            25     1048611     0x00100023
branch                 100     1048612     0x00100024
teller                 101     1048613     0x00100025
account                102           0     0x00000000
history                103     1048615     0x00100027
results                104     1048616     0x00100028
```

Figure 38-9
Output from systables Query Showing partnum Values

To obtain the actual tblspace numbers for the fragments that make up the table, you must query the **sysfragments** table for the same database. Figure 38-10 shows that the **account** table from Figure 38-9 on page 38-21 has three table fragments and three index fragments.

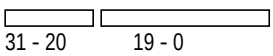
```
tabid fragtype          partn       hex_tblspace_name

  102 T                 1048614     0x00100026
  102 T                 2097154     0x00200002
  102 T                 3145730     0x00300002
  102 I                 1048617     0x00100029
  102 I                 2097155     0x00200003
  102 I                 3145731     0x00300003
```

Figure 38-10
Output from sysfragments Table with partn Values

TblSpace Number Elements

The hexadecimal representation of the tblspace number is a composite of two numbers. The most-significant 12 bits ($1\frac{1}{2}$ bytes) indicate the dbspace number where the tblspace resides. The least-significant 20 bits ($2\frac{1}{2}$ bytes) indicate the logical page number where the tblspace is described. Figure 38-11 illustrates the elements of a tblspace number. Compare this format to the actual **partnum** values in Figure 38-9 on page 38-21.

The first page in a tblspace is logical page 0. (Physical page numbers refer to the address of the page in the chunk.) For example, the tblspace number of the tblspace tblspace in the root dbspace is always 0x1000001. The root space tblspace tblspace is always contained in the first dbspace and on logical page 1 within the tblspace tblspace. (The bit-map page is page 0.)



0 - 19 page number within the tblspace tblspace
20 - 31 dbspace number

Figure 38-11
The TblSpace Number with Dbspace Number and Page Number

TbIspace TbIspace Size

The size of the tblspace tblspace is always 50 pages. These tblspace tblspace pages are allocated as an extent when the dbspace is initialized. If the database server attempts to create a table, but the tblspace tblspace is full, the database server allocates a next extent to the tblspace.

When a table is removed from the dbspace, its corresponding entry in the tblspace tblspace is deleted. The space in the tblspace is released and can be used by a new tblspace.

TbIspace TbIspace Bit-Map Page

The first page of the tblspace tblspace, like the first page of any initial extent, is a bit map that describes the page fullness of the following pages. Each page that follows has an entry on the bit-map page. If needed, additional bit-map pages are located throughout the contiguous space allocated for the tblspace, arranged so that each bit map describes only the pages that follow it, until the next bit map or the end of the dbspace. Bit-map pages fall at distinct intervals within tblspaces pages. Each bit-map page describes a fixed number of pages that follow it.

Structure of the Database Tblspace

The database tblspace appears only in the initial chunk of the root dbspace. The database tblspace contains one entry for each database managed by the database server. Figure 38-12 illustrates the location of the database tblspace.

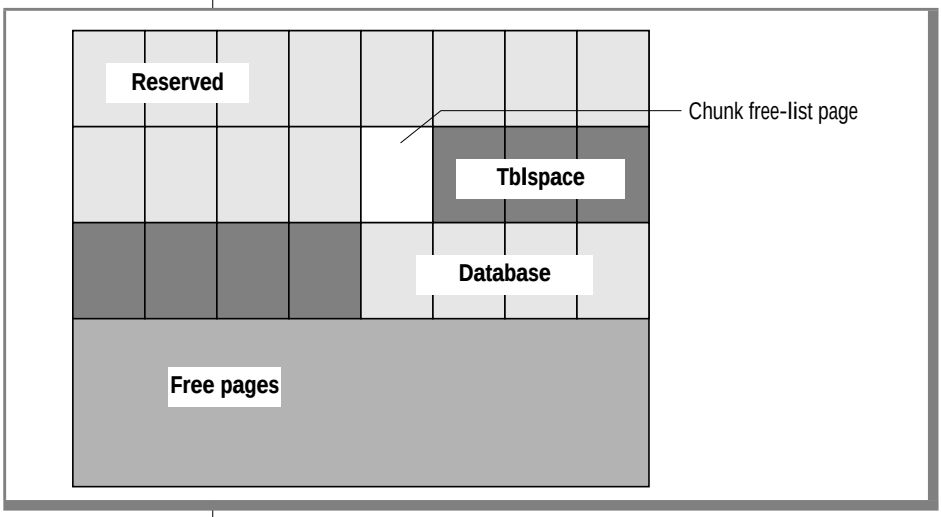


Figure 38-12
Database Tblspace Location in Initial Chunk of Root Dbspace

What Is the Database Tblspace Number?

The tblspace number of the database tblspace is always 0x100002. This tblspace number appears in an **onstat -t** listing if the database tblspace is active.

Database Tblspace Entries

Each database tblspace entry includes the following five components:

- Database name
- Database owner
- Date and time that the database was created
- The tblspace number of the **systables** system catalog table for this database
- Flags that indicate logging mode

The database tblspace includes a unique index on the database name to ensure that every database is uniquely named. For any database, the **systables** table describes each permanent table in the database. Therefore, the database tblspace only points to the detailed database information located elsewhere.

When the root dbspace is initialized, the database tblspace first extent is allocated. The initial-extent size and the next-extent size for the database tblspace are four pages. You cannot modify these values.

Structure of a Dbspace Bit-Map Page

Extents contain one or more bit-map pages that track free pages in the extent. Each bit-map entry describes the fullness of one page in the extent. The number of bit-map pages needed for an extent depends on three variables:

- Number of pages in the extent, which affects the number of bit-map entries needed
- Page size, which affects the number of bit-map entries that can fit on a page
- Type of the bit-map entries, which depends on the type of data stored on the page

Types of Bit-Map Entries

All bit-map pages are initialized and linked when the extent is allocated. The bit-map pages are scattered throughout the extent. The first page in the extent, and every (*n* +1)th page thereafter, is designated as a bit-map page, where *n* is the number of bit-map entries that fit on a single page. The pages described by a bit-map page can span extents.

The database server uses two types of bit-map pages, a two-bit bit-map page (which contains two-bit entries) and a four-bit bit-map page (which contains four-bit entries).

Two-Bit Bit-Map Pages

The two-bit bit-map pages track available space in extents allocated to tables that meet two criteria:

- The table contains fixed-length rows that are smaller than a page.
- The table does not contain VARCHAR, BYTE, or TEXT data types.

Only two bits are needed to describe page fullness for these limited conditions, as illustrated in Figure 38-13.

Figure 38-13
Bit Values for Two-Bit Bit-Map Pages

Bit Values	Description of Page Fullness
00	Page is unused.
01	Page is partially used (data page).
10	Page is used completely (index page).
11	Page is full (data page).

Four-Bit Bit-Map Pages

The four-bit bit-map pages track available space in extents allocated to tables that contain rows longer than a page, or rows that include VARCHAR, BYTE, or TEXT data types. Four bits are needed to describe all possible combinations of page fullness for these extents, as illustrated in Figure 38-14. The terms used to describe page fullness describe row segments as whole-page, partial-page, and small. These segment sizes are relative to available free space and are selected on the basis of performance.

Figure 38-14
Four-Bit Bit-Map Values

Bit Values	Description of Page Fullness
0000	Page is unused.
0100	Home data page has room for another data row.
1000	Page is used completely (index page).
1100	Home data page is full.
0001	Remainder page, can accept whole-page segments
0101	Remainder page, room for partial-page segments
1001	Remainder page, room for small segments
1101	Remainder page, no room for even small segments
0010	Blob page, can accept whole-page segments
0110	Blob page, room for partial-page segments
1010	Blob page, room for small segments
1110	Blob page, no room for even small segments

Structure and Allocation of an Extent

This section covers the following topics:

- What an extent is
- How extent size can vary
- The three types of pages contained in an extent
- How a new extent is allocated
- Automatic doubling of next-extent size
- What the database server does when it cannot find sufficient contiguous space for an extent
- Extent merging

Extent Structure

An extent is a collection of contiguous pages within a dbspace. 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 every extent thereafter, becomes full.

Blobspaces do not employ the concept of an extent.

For specific instructions on how to specify and calculate the size of an extent, see your *Performance Guide*.

Extent Size

The minimum size of an extent is four pages. The default size of an extent is eight pages. No maximum limit exists, although a practical limit is about two gigabytes (or as much space as is available within the chunk). The maximum size of an extent is determined by the largest page number that can be accommodated in a rowid. Since the page number in a rowid cannot exceed 16,777,215, this figure is the upper limit of the number of pages that a single extent can contain.

Tblspaces that hold *index fragments* follow different rules for extent size. The database server bases the extent size for these tblspaces on the extent size for the corresponding table fragment. The database server uses the ratio of the row size to index key size to assign an appropriate extent size for the detached index tblspace. (For information on fragmented indexes, see "Fragmenting Table Indexes" on page 15-8.)

Tblspaces that hold *index fragments* follow different rules for extent size. The database server bases the extent size for these tblspaces on the extent size for the corresponding table fragment. The database server uses the ratio of the row size to index key size to assign an appropriate extent size for the index tblspace. (For information on fragmented indexes, see "Fragmenting Table Indexes" on page 15-8.)

Page Types Within an Extent

Within the extent, individual pages contain different types of data. Extent pages can be separated into five categories:

- Data pages

  Data pages contain the data rows for the table.

- Index pages (root, branch, and leaf pages)

  Index pages contain the index information for the table.

- Bit-map pages

  Bit-map pages contain control information that monitors the fullness of every page in the extent. If the table contains a VARCHAR, BYTE, or TEXT data type, or if the length of one row is greater than a page, the bit map is a four-bit bit map; otherwise, the bit map is a two-bit bit map.

- Blobpages

  Blobpages contain TEXT and BYTE data types that are stored with the data rows in the dbspace. TEXT and BYTE data types that reside in a blobspace are stored in blobpages, a structure that is completely different than the structure of a dbspace blobpage.

- Free pages

  Free pages are pages in the extent that are allocated for tblspace use, but whose function has not yet been defined. Free pages can be used to store any kind of information: data, index, TEXT or BYTE data types, or bit map.

Important: *An extent that is allocated for a table fragment does not contain index pages. Index pages for a fragmented table always reside in a separate tblspace. (For information on fragmented indexes, see "Fragmenting Table Indexes" on page 15-8.)*

Figure 38-15 on page 38-30 illustrates the possible structure of a nonfragmented table with an initial-extent size of 8 pages and a next-extent size of 16 pages.

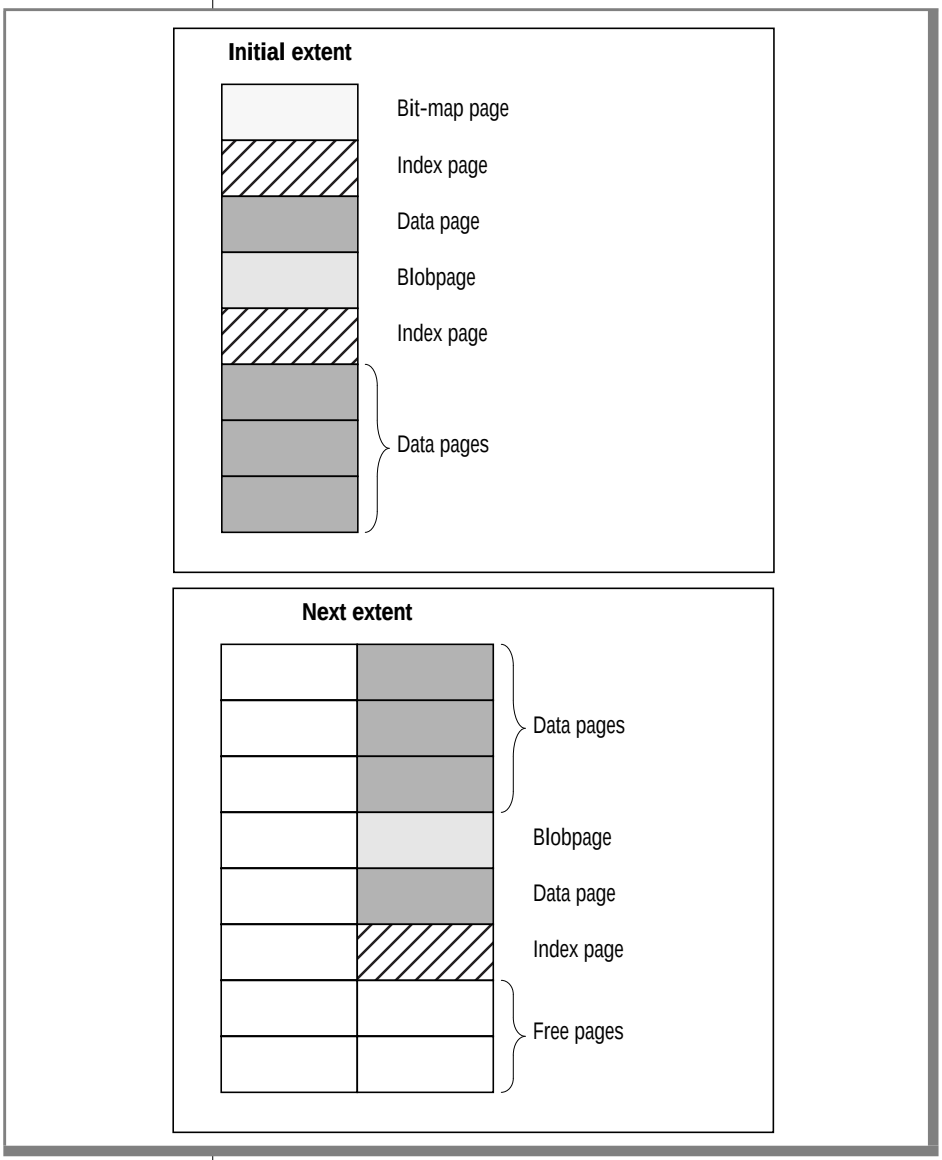


Figure 38-15
Extent Structure of a Table

Next-Extent Allocation

After the initial extent fills, the database server attempts to allocate another extent of contiguous disk space. The procedure that the database server follows is referred to as next-extent allocation.

Extents for a tblspace are tracked as one component of the tblspace tblspace information for the table. The maximum number of extents allocated for any tblspace is application and machine dependent because it varies with the amount of space available on the tblspace tblspace entry.

Next-Extent Size

The number of kilobytes that the database server allocates for a next extent is, in general, equal to the size of a next extent, as specified in the SQL statement CREATE TABLE. However, the actual size of the next-extent allocation might deviate from the specified size because the allocation procedure takes into account the following three factors:

- Number of existing extents for this tblspace
- Availability of contiguous space in the chunk and dbspace
- Location of existing tblspace extents

The effect of each of these factors on next-extent allocation is explained in the paragraphs that follow and in Figure 38-16 on page 38-33.

Extent Size Doubles Every 16 Extents

If a tblspace already has 16 extents allocated, the database server automatically doubles the size for subsequent allocations. This doubling occurs every 16 extents. For example, if you create a table with NEXT SIZE equal to 20 kilobytes, the database server allocates the first 16 extents at a size of 20 kilobytes each. The database server allocates extents 17 to 32 at 40 kilobytes each, extents 33 to 48 at 80 kilobytes each, and so on.

What If the Database Server Cannot Find Contiguous Space?

If the database server cannot find available contiguous space in the first chunk equal to the size specified for the next extent, it extends the search into the next chunk in the dbspace. Extents are not allowed to span chunks.

If the database server cannot find adequate contiguous space anywhere in the dbspace, it allocates to the table the largest available amount of contiguous space. (The minimum allocation is four pages. The default value is eight pages.) No error message is returned if an allocation is possible, even when the amount of space allocated is less than the requested amount.

Extents for the Same Table Might Merge

If the disk space allocated for a next extent is physically contiguous with disk space already allocated to the same table, the database server allocates the disk space but does not consider the new allocation as a separate extent. Instead, the database server extends the size of the existing contiguous extent. Thereafter, all disk-space reports reflect the allocation as an extension of the existing extent. That is, the number of extents reported is always the number of physically distinct extents, not the number of times a next extent has been allocated plus one (the initial extent). Figure 38-16 on page 38-33 illustrates extent-allocation strategies.

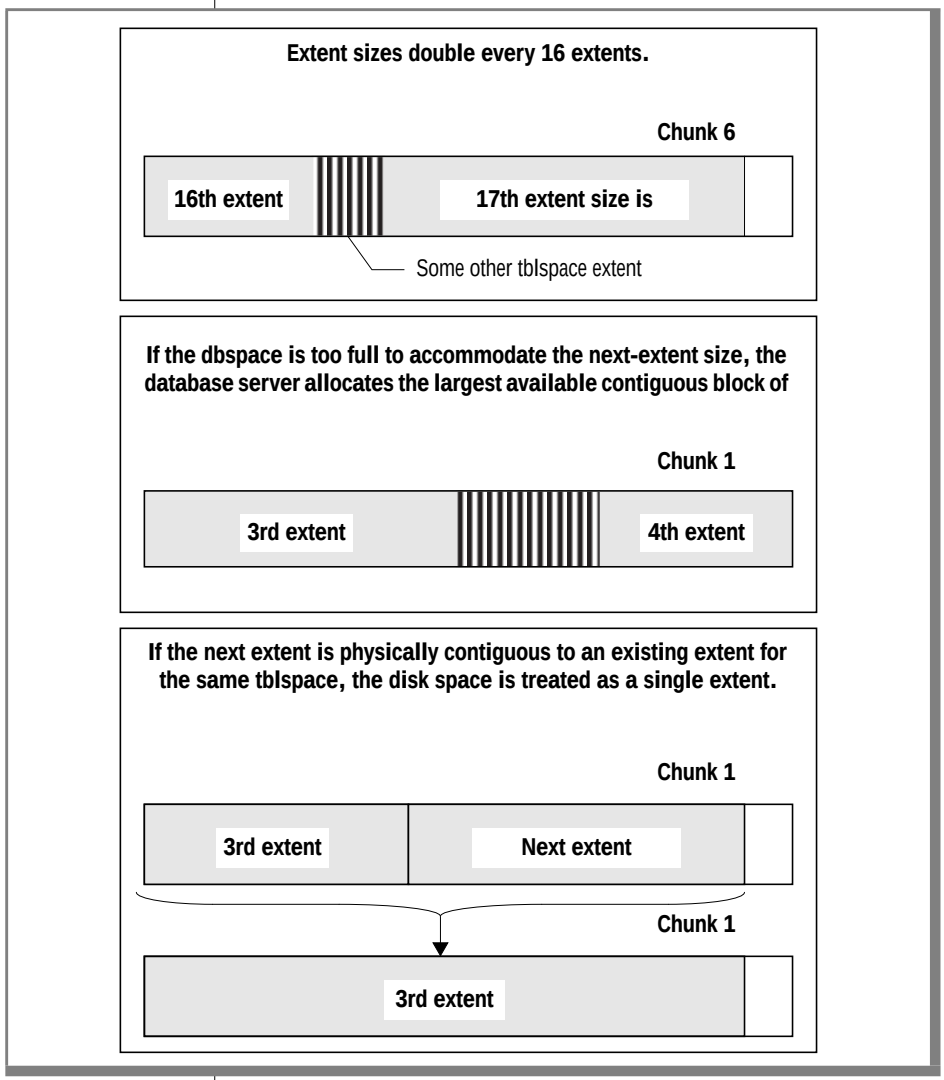


Figure 38-16
Next-Extent Allocation Strategies

After disk space is allocated to a tblspace as part of an extent, the space remains dedicated to that tblspace even if the data contained in it is deleted. For alternative methods of reclaiming this empty disk space, see your *Performance Guide*.

Structure and Storage of a Dbspace Page

This section describes how a dbspace page is structured and how the database server uses this structure to store data in a dbspace.

Structure of a Dbspace Page

The basic unit of database-server I/O is a page. Page size might vary among computers. You cannot modify the page size.

Page Elements

The database server allocates pages in a group called an extent. Pages can be categorized according to the type of information they contain. All pages managed by the database server adhere to a similar structure, although the function of the page can alter slightly the size of structures within the page. Figure 38-17 illustrates the following three structures that appear on every page:

- Page header (24 bytes, including one 4-byte time stamp)
- Page-ending time stamp (4 bytes)
- Slot table (4 bytes per entry)

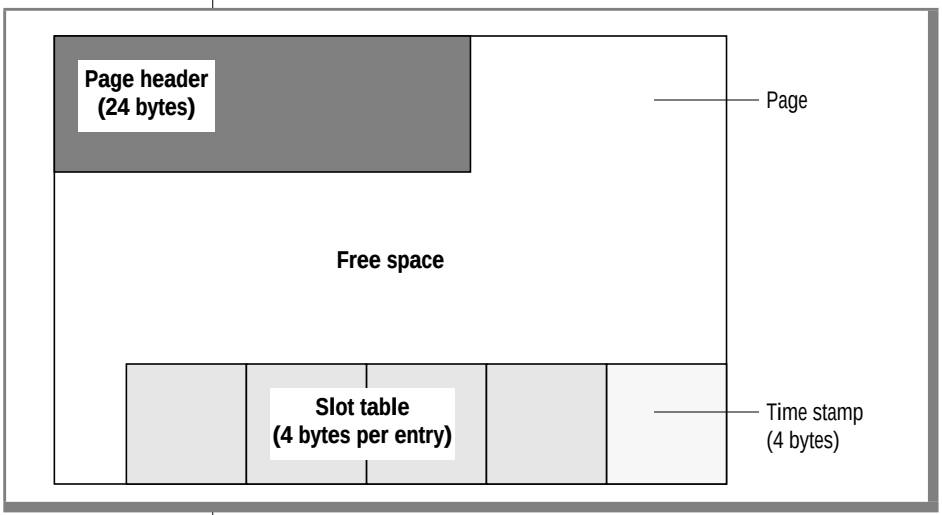


Figure 38-17
Dbspace Page Structure

Page Headers

The page header includes the following six components:

- Page-identification number (address of the page on disk)
- Number of slot-table entries used on the page (used to calculate where to locate the next slot-table entry)
- Number of free bytes left on the page
- Pointer to the contiguous free space on the page that lies between the last data entry and the first slot-table entry
- Time stamp that changes each time the page contents are modified
- Two index-related pointers (used if the page is used as an index page)

Page Time Stamps

The page-header time stamp and the page-ending time stamp function as a pair to validate page consistency. Each time the page contents are modified, a time stamp is placed in the page header. At the end of the write, the header time stamp is copied into the last four bytes on the page. Subsequent access to the page checks both time stamps. If the two time stamps differ, this inconsistency is reported as a part of consistency checking.

Page Slot Tables

The slot table is a string of 4-byte slot-table entries that begins at the page-ending time stamp and grows toward the beginning of the page. The entries in the slot table enable the database-server user processes to find data on dbspace pages. Each entry in the slot table describes one segment of data that is stored in the page. The number of the slot-table entry is stored as a 1-byte unsigned integer. The slot-table entries cannot exceed 255. This number is the upper limit on the number of rows, or parts of a row, that can be stored in a single data page.

The slot-table entry is composed of the following two parts:

- Page offset where the data segment begins
- Length of the data segment

For example, in a data page, the slot-table entry would describe the page offset where the data row (or portion of a data row) starts and the length of the row (or portion of a row).

Slot Table and Rowid

The number of the slot-table entry is stored as part of the data-row rowid. The data-row rowid is a unique identifier for each data row. It is composed of the page number where the row is stored and the number of the slot-table entry that points to that data row.

As part of a rowid, the number of the slot-table entry is stored as a 1-byte unsigned integer. Since the rowid cannot store a slot-table entry greater than 255, this number is the upper limit on the number of rows than can be stored in a single data page.

Slot Table Stays Accurate Even If Row Moves

The slot table is the only database-server structure that points to a specific location in a data page. For this reason, the database server can initiate page compression whenever required, according to internal algorithms. Typically, page compression changes the location of the data row in the page and, therefore, generates a new page offset that is written into the slot-table entry. However, the number of the slot-table entry remains fixed. Thus all forwarding pointers and descriptor values that rely on a rowid value remain accurate.

Rowid in Nonfragmented Tables

The database server can store rows that are longer than a page. The database server also supports the VARCHAR data type, which results in rows of varying length.

As a result, rows do not conform to a single format. The following facts about rows must be considered when the database server stores data rows in a page:

- Rows within a table are not necessarily the same length if the table contains one or more columns of type VARCHAR. In addition, the length of a row in such a table might change when an end user modifies data contained in the VARCHAR column.
- The length of a row can be greater than a page.
- TEXT and BYTE data types are not stored within the data row. Instead, the data row contains a 56-byte descriptor that points to the location of the data. (The descriptor can point to either a dbspace page or a blobspace blobpage. If you are using Optical Subsystem, the descriptor can also point to an optical-storage subsystem.)

For instructions about how to estimate the length of fixed-length and variable-length data rows, see your *Performance Guide*.

What Is a Rowid?

In the context of a nonfragmented table, the term *rowid* refers to a unique 4-byte integer that is a function of the physical location of the row in the table. Rowid is the combination of a page identification number (the logical page number) and the number of an entry in the slot table on that page. The rowid defines the location of a data row. The page that contains the first byte of the data row is the page that is specified by the rowid. This page is called the data row *home page*.

Fragmented tables can also have rowids, but they are implemented in a different way. For more information on this topic, see "Rowid in Fragmented Tables" on page 38-39.

Rowids Never Change

The rowid structure permits the length of the row and its location on a page to change without affecting the contents of the rowid. Either change, a change in length caused by an insert or a delete or a change in location on the page caused by page compression, is reflected in the entry stored in the slot table. If the page where the data row is stored changes, a forward pointer is left on the home page. In all cases, the rowid remains unchanged. Figure 38-18 illustrates the rowid format.

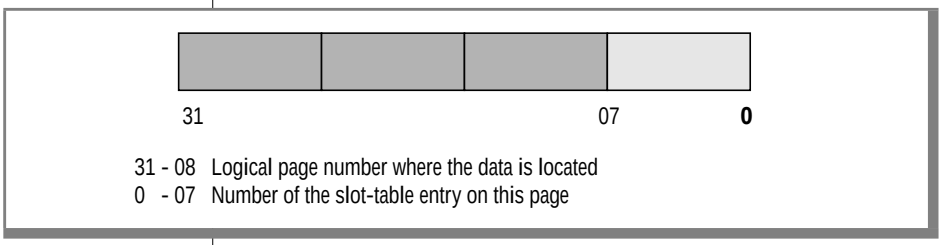


Figure 38-18
Rowid Format

Logical Page Number

The logical page number describes the data row home page. The logical page number is stored in the most-significant three bytes of the rowid as an unsigned integer. Logical pages are numbered relative to the tblspace. That is, the first logical page in a tblspace is page 0. (Physical page numbers refer to the address of the page in the chunk.) For example, if you create a table, and the resulting initial extent is located in the middle of a chunk, the physical address of the first page in the extent represents the location in the chunk. The logical address for the same page is zero. Since the largest number that can be stored in the three most-significant bytes of a rowid is 16,777,215, this figure is the upper limit of the number of pages that can be contained in a single tblspace.

How Does the Database Server Use Rowid?

Every data row in a nonfragmented table is uniquely identified by an unchanging rowid. When you create an index for a nonfragmented table, the rowid is stored in the index pages associated with the table to which the data row belongs. When the database server requires a data row, it searches the index to find the key value and uses the corresponding rowid to locate the requested row. If the table is not indexed, the database server might sequentially read all the rows in the table.

Eventually, a row might outgrow its original storage location. If this occurs, a *forward pointer* to the new location of the data row is left at the position defined by the rowid. The forward pointer is itself a rowid that defines the page and the location on the page where the data row is now stored.

Rowid in Fragmented Tables

Unlike rows in a nonfragmented table, the database server does *not* assign a rowid to rows in fragmented tables. If you wish to access data by rowid, you must explicitly create a rowid column as described in the *Performance Guide for Informix Dynamic Server*. If user applications attempt to reference a rowid in a fragmented table that does not contain a rowid that you explicitly created, the database server returns an appropriate error code to the application.

Accessing Data Using Rowid in Fragmented Tables

From the viewpoint of an application, the functionality of a rowid column in a fragmented table is identical to the rowid of a nonfragmented table. However, unlike the rowid of a nonfragmented table, the database server uses an index to map the rowid to a physical location.

When the database server accesses a row in a fragmented table using the rowid column, it uses this index to look up the physical address of the row before it attempts to access the row. For a nonfragmented table, the database server uses direct physical access without having to do an index lookup. As a consequence, accessing a row in a fragmented table using rowid takes slightly longer than accessing a row using rowid in a nonfragmented table. You should also expect a small performance impact on the processing of inserts and deletes due to the cost of maintaining the rowid index for fragmented tables.

Primary-key access can lead to significantly improved performance in many situations, particularly when access is in parallel.

Informix Recommendations on Use of Rowid

Informix recommends that application developers use primary keys as a method of access rather than rowids. Because primary keys are defined in the ANSI specification of SQL, using them to access data makes your applications more portable.

For a complete description on how to define and use primary keys to access data, see the *Informix Guide to SQL: Reference* and the *Informix Guide to SQL: Tutorial*.

Data-Row Format and Storage

The variable length of a data row has the following consequences for row storage:

- A page might contain one or more whole rows.
- A page might contain portions of one or more rows.
- A page might contain a combination of whole rows and partial rows.
- An updated row might increase in size and become too long to return to its original storage location in a row.

The following paragraphs describe the guidelines that the database server follows during data storage.

How Are Rows Stored?

To minimize retrieval time, rows are not broken across page boundaries unnecessarily. Rows that are shorter than a page are always stored as whole rows. A page is considered *full* when the count of free bytes is less than the number of bytes needed to store a row of maximum size. Figure 38-19 illustrates data storage when rows are less than a page.

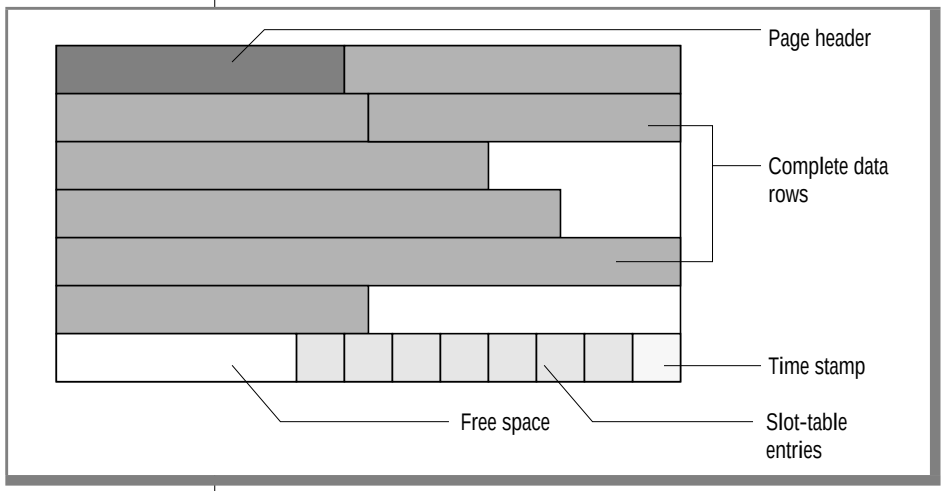


Figure 38-19
Rows Shorter Than a Page

Where Are Rows Stored?

When the database server receives a row that is longer than a page, the row is stored in as many whole pages as required. The database server then stores the trailing portion in less than a full page.

The page that contains the first byte of the row is the row home page. The number of the home page becomes the logical page number contained in the rowid. Each full page that follows the home page is referred to as a big-remainder page. If the trailing portion of the row is less than a full page, it is stored on a remainder page.

After the database server creates a remainder page to accommodate a long row, it can use the remaining space in this page to store other rows.

Figure 38-20 illustrates the concepts of home page, big-remainder page, and remainder page.

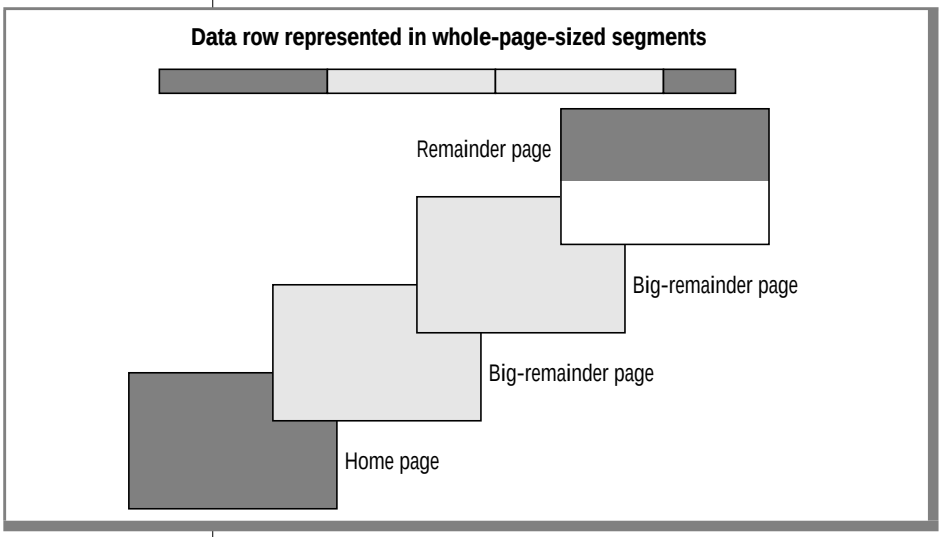


Figure 38-20
Remainder Pages

How Are Data Row Sections Linked?

When a row is longer than one page, but less than two pages, the home row contains a forward pointer to a remainder page. The forward pointer is always stored as the first 4 bytes in the data portion of the page. The forward pointer contains the rowid of the next portion of the row. A flag value is added to the slot-table entry of the data row to indicate that a pointer exists.

When a row is longer than two pages, the home row and each big-remainder page contain forward pointers to the next portion of the data row. Figures 38-21 through illustrates data storage for rows that are longer than two pages

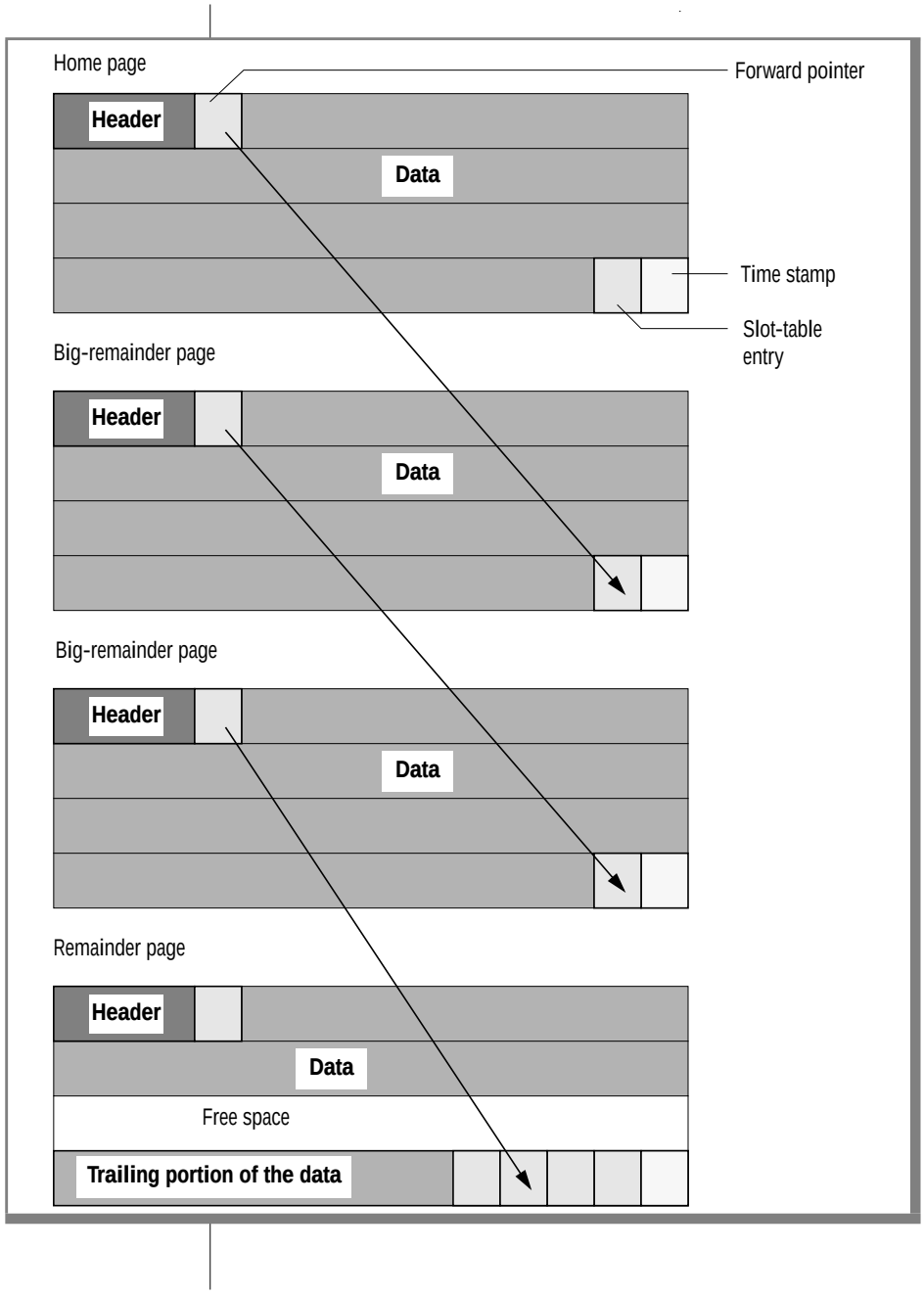


Figure 38-21
Rows Longer Than Two Pages

What Happens When a Row Is Modified?

When a row is modified, the database server attempts to return the modified row to its current location. If the row size is unchanged, no changes are needed in the slot table. If the row is smaller than before, the database server changes the slot-table entry for this row to reflect the new row length. If the row no longer fits, the database server attempts to store the row in another location on the same page. If the database server can do this, the slot-table entry is changed to reflect both the new starting offset and the new length of the row.

What Happens If the Updated Row Is Too Large for the Home Page?

If the modified data row is shorter than a page but cannot be accommodated on the current page, a 4-byte forwarding pointer (that contains the new rowid) is stored on the home page. The data row retains its original rowid, which is stored in the index page. The data is moved to the new page, and the space freed by the move is available for other rows. Figure 38-22 on page 38-45 illustrates data storage if the updated row is too large for the home page but shorter than a whole page.

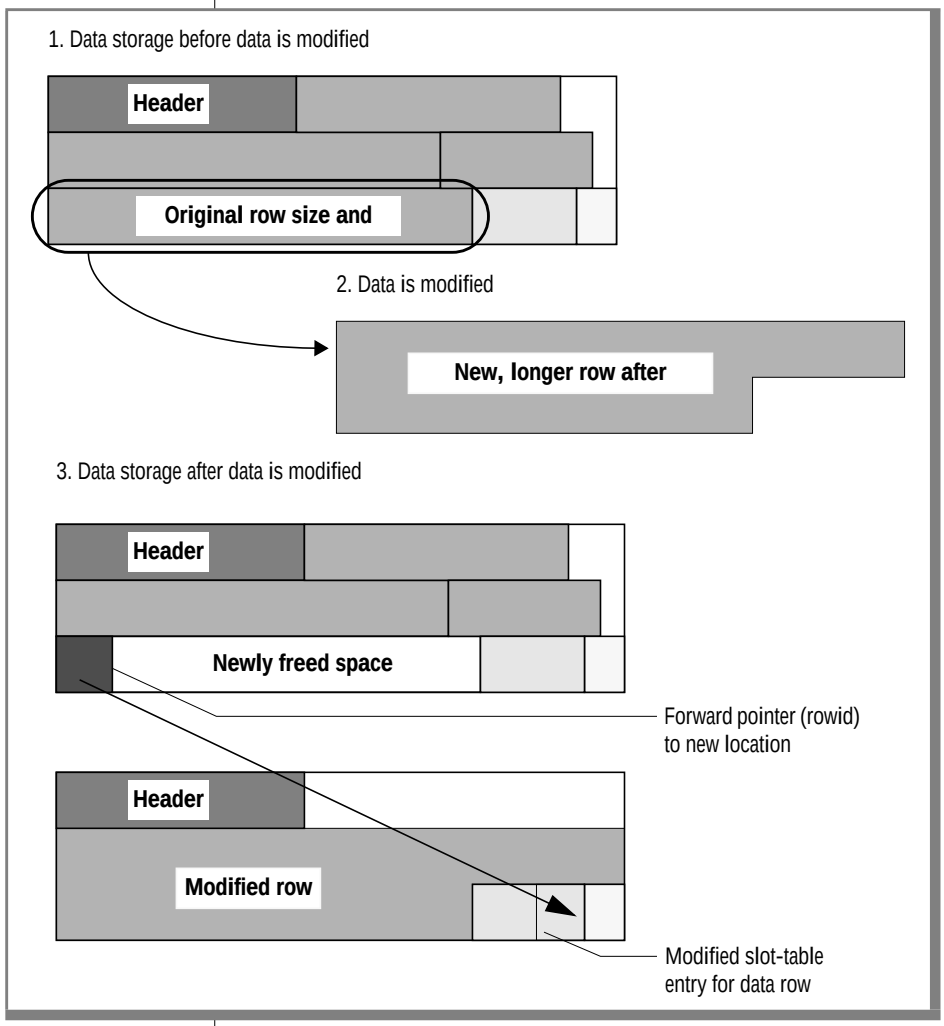


Figure 38-22
Updated Rows

What Happens When the Updated Row Is Longer Than a Whole Page?

If the modified data row is longer than a page, the database server first begins to divide the data into whole-page segments, starting from the *tail* end of the row. Then the database server tries to fit the leading segment plus 4 bytes (for the forward pointer) into the current location of the row on the home page. If the leading segment fits, the whole-page tail segments are stored in big-remainder pages, and forwarding pointers are added.

If the leading segment cannot fit into the current location of the row on the home page, the database server divides the page into whole-page segments again, this time beginning with the leading end of the row. The database server stores only a forwarding pointer in the current page location. The rest of the data row is stored in whole-page segments on one or more big-remainder pages. Forward pointers are added to each page. The trailing portion of the row is stored on a remainder page. Figure 38-23 on page 38-47 illustrates storage of an updated row that is longer than a whole page.

Page Compression

Over time, the free space on a page can become fragmented. When the database server attempts to store data, it first checks row length against the number of free bytes on a page to determine if the row fits. If adequate space is available, the database server checks if the page contains adequate contiguous free space to hold the row (or row portion). If the free space is not contiguous, the database server calls for page compression.

During page compression, a user process locates a free buffer in the shared-memory buffer pool and copies to the buffer the data-page header and page time stamp. Then, starting from the first slot-table entry, the user process copies each slot-table entry and its associated data, updating the slot-table information as the data is written to the buffer. When the process completes the rewriting, the newly compressed page is written back to the data page.

All free space in the data page is now contiguous, and the row (or row portion) is stored according to the usual procedure of writing the data and its associated slot-table entry.

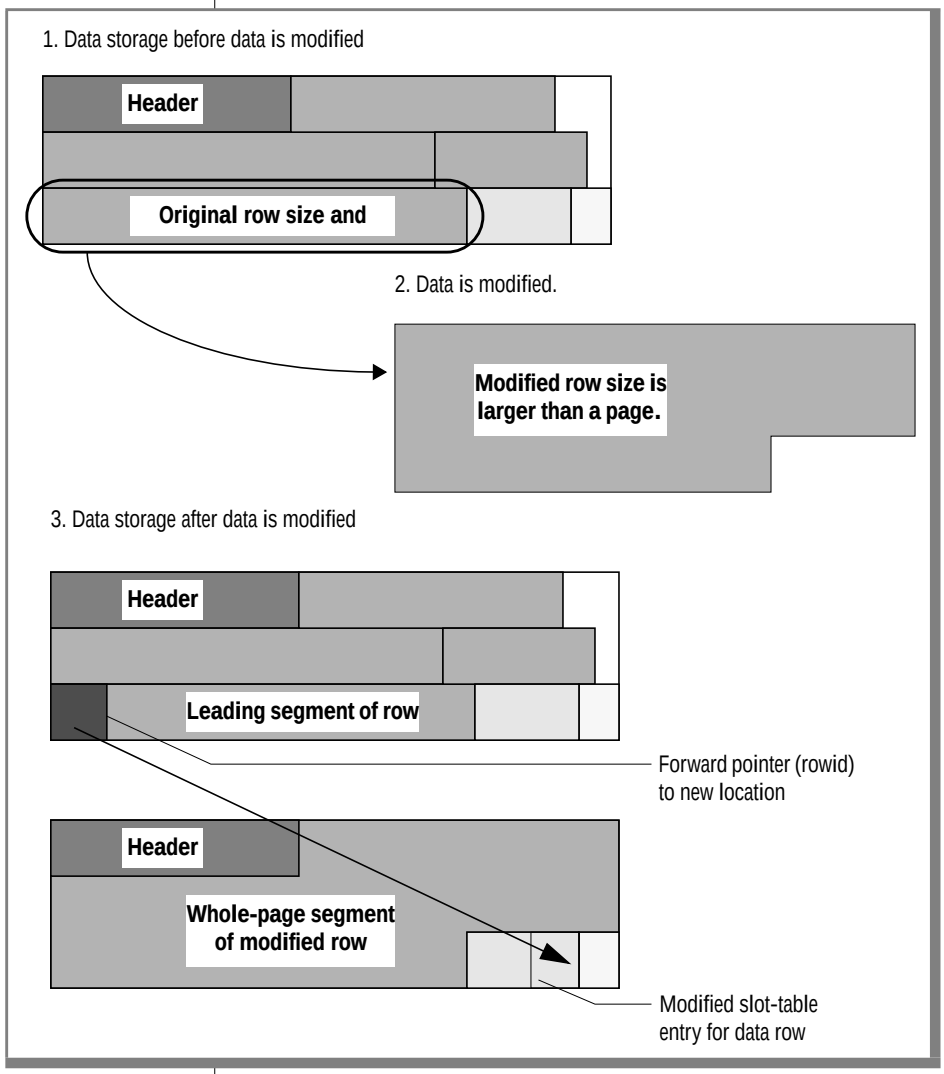


Figure 38-23
Updated Rows That Span Pages

Structure of B-Tree Index Pages

This section provides general information about the structure of B-tree index pages. It is designed as an overview for the interested reader; it does not describe all the rules governing index creation, page splitting, and page merging.

The section begins with a definition of terms used in B-tree indexing. An overview of the process that the database server uses to create and fill indexes is then explained. The latter part of the section describes the physical layout of an index page.

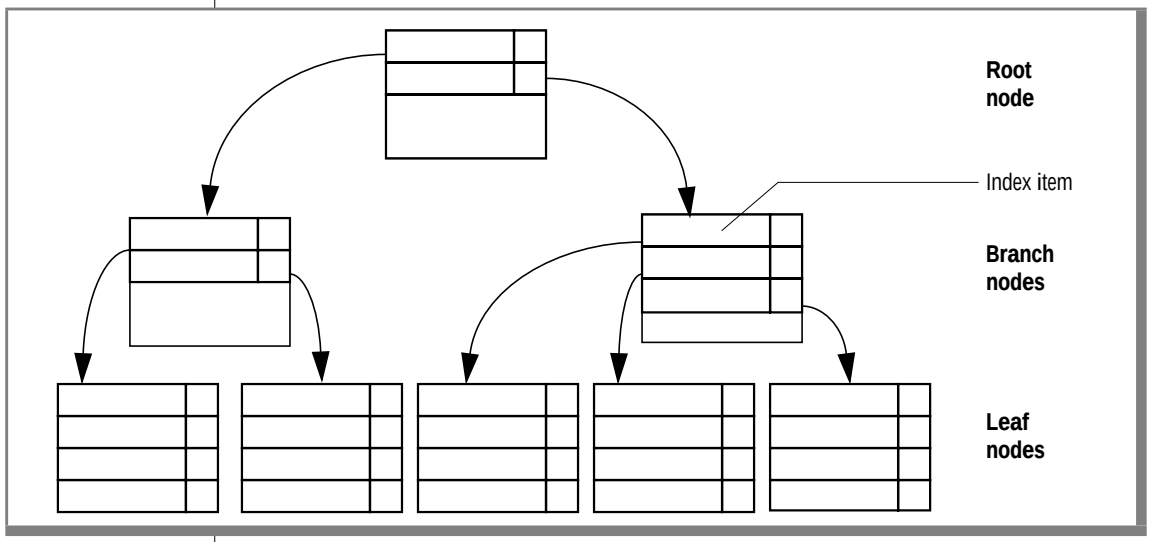
Definition of Terms in B-Tree Indexing

The database server uses a B-tree structure for organizing index information. As shown in Figure 38-24 on page 38-49, a fully developed index is composed of the following three different types of index pages, or nodes:

- One *root node*
- Two or more *branch nodes*
- Many *leaf nodes*

Each node serves a different function. The following sections describe each node and the role that it plays in indexing.

Figure 38-24
Full B-Tree Structure



What Is an Index Item?

The fundamental unit of indexes is the *index item*. As shown in Figure 38-25, an index item is made up of a *key value*, a *rowid*, and a *delete* flag. A key value represents the value of the indexed column for a particular row. A rowid is a unique identifier for the row that indicates its location in the table. A delete flag is a one-byte indicator that specifies whether the row that the rowid points to still exists or has been deleted.

A delete flag value of `0` indicates the row still exists; a value of `1` indicates the row has been deleted. When a transaction deletes an item, the database server locks the item and sets the delete flag to `1`. Items with delete flags that indicate deleted rows are purged from the B-tree structure frequently. Only leaf nodes contain items with delete flags.

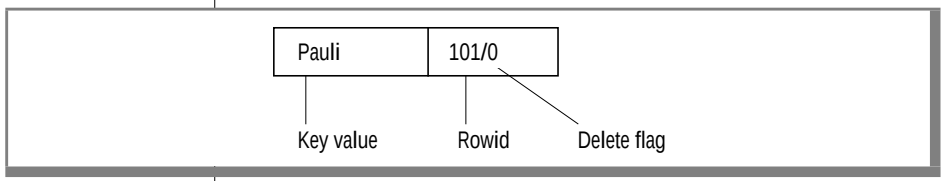


Figure 38-25
Logical Depiction of an Index Item

Figure 38-25 on page 38-49 illustrates an example of an index item. The index item illustrated is row 101 in the **customer** table. The **lname** value of row 101 is `Pauli`. The delete flag is set to `0`, indicating that the row still exists. The index item for this **lname** value is composed of a key value, `Pauli`, a rowid for data row 101, and a delete flag, `0`.

What Is a Node?

A node is an index page that stores a group of index items. Figure 38-26 shows a logical depiction of an index node.

Albertson	207/0
Baxter	208/0
Beatty	206/0
Currie	201/0
Keyes	204/0
Lawson	205/0
Mueller	203/0
Wallach	209/0
Watson	202/0

Figure 38-26
An Index Node

B-tree structures contain the following three types of nodes: root, branch, and leaf nodes. When an index is fully developed (see Figure 38-24 on page 38-49), the three types of nodes can be described as follows:

- A root node contains node pointers to branch nodes.
- A branch node contains pointers to leaf nodes or other branch nodes.
- A leaf node contains index items and horizontal pointers to other leaf nodes.

Although not necessary for the explanation of indexing presented here, a node contains other elements that are described in "Physical Storage Format of Index Pages" on page 38-55.

Logical Storage of Indexes

This section presents an overview of how the database server creates and fills an index. For clarity, the physical storage details of indexing are left out of this section. For this level of detail, see "Physical Storage Format of Index Pages" on page 38-55.

Creation of Root and Leaf Nodes

When you create an index for an empty table, the database server allocates a single index page. This page represents the root node and remains empty until you insert data into the table.

At first, the root node functions like a leaf node. For each row that you insert into the table, the database server creates and inserts an index item into the root node. Figure 38-27 illustrates how a root node appears before it fills.

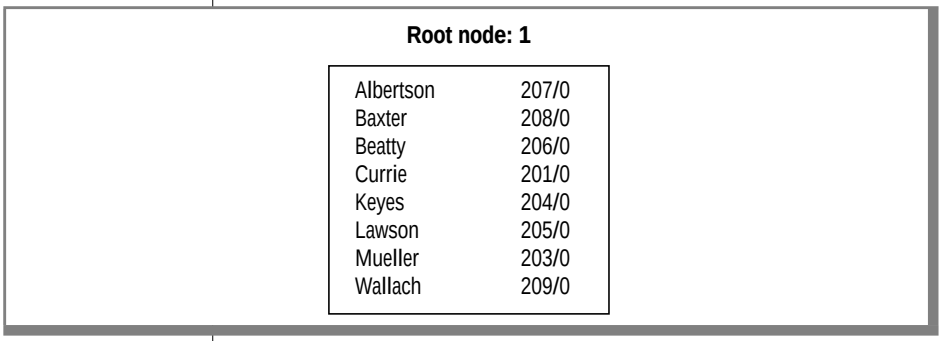


Figure 38-27
Root Node

When the root node becomes full, the database server first creates two leaf nodes. The database server then moves approximately half of the root-node entries to each of the newly created leaf nodes, as depicted in Figure 38-28.

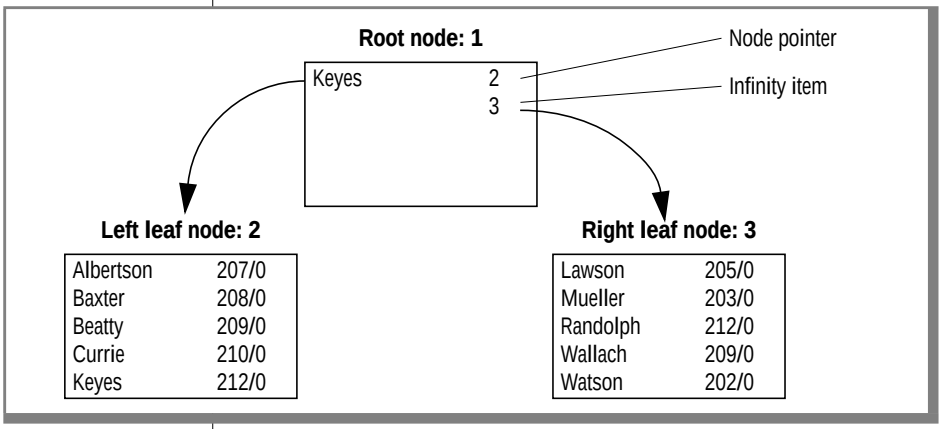


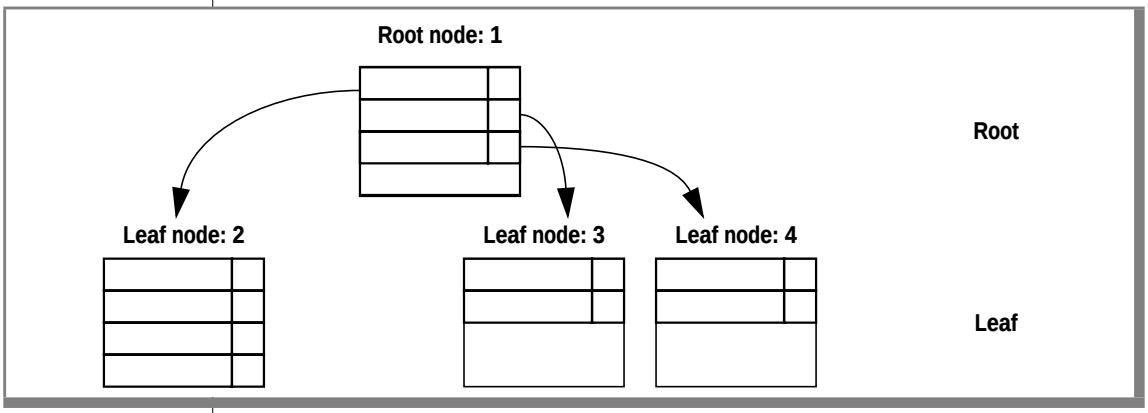
Figure 38-28
Two Leaf Nodes Created After a Root Node Becomes Full

The root node retains only two entries. The first entry consists of a key value plus a pointer to a leaf node. The key value in the root node is the same as that of the last index item of the left branch node. The second item is called an *infinity item*. An infinity item has no key value but instead contains a pointer that points to the rightmost leaf node.

As you add new rows to a table, the database server adds index items to the leaf nodes. When a leaf node fills, a new leaf node is created, and the database server moves part of the contents of the full node to the new node. A node pointer to the new leaf node is added to the root node.

For example, suppose that leaf node 3 in Figure 38-28 on page 38-51 becomes full. When this occurs, the database server adds yet another leaf node. The database server moves part of the records of leaf node 3 to the new leaf node as depicted in Figure 38-29.

Figure 38-29
Leaf Node 4 Created After Leaf Node 3 Fills

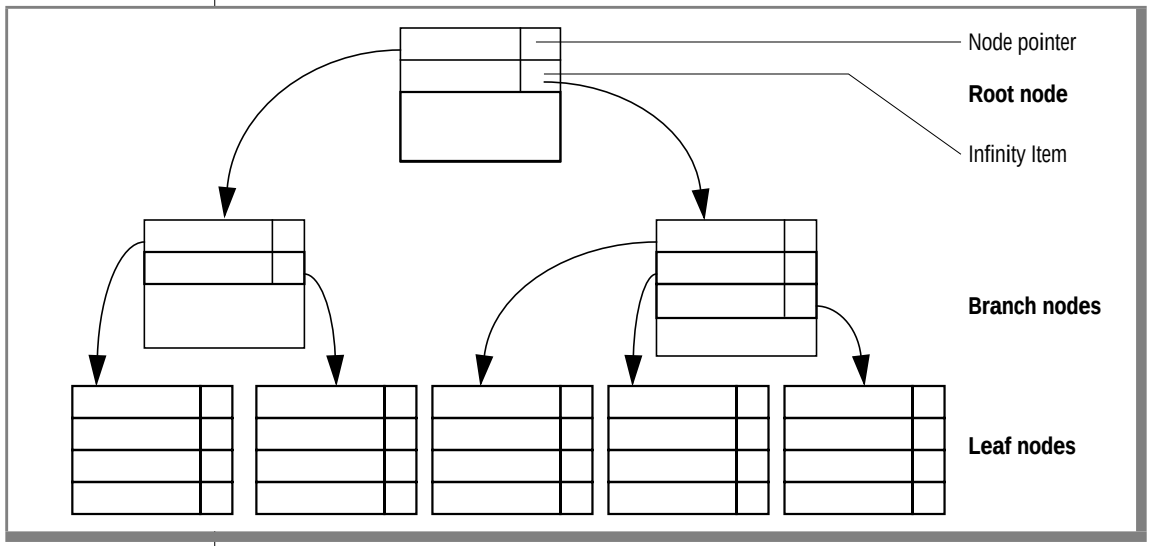


Creation of Branch Nodes

Eventually, as you add rows to the table, the database server fills the root node with node pointers to all the existing leaf nodes. When the database server splits yet another leaf node, and the root node has no room for an additional node pointer, the following process occurs.

The database server splits the root node and divides its contents among two newly created branch nodes. Just as it did in its first split, which created the leaf level, the root node retains only two entries: one node pointer to the left branch node and an infinity item pointing to the right one. Figure 38-30 illustrates this structure.

Figure 38-30
Full B-Tree Structure



As index items are added, more and more leaf nodes are split, causing the database server to add more branch nodes. Eventually, the root node fills with pointers to these branch nodes. When this occurs, the database server splits the root node again. The database server then creates yet another branch level between the root node and the lower branch level. This process results in a 4-level tree, with one root node, two branch levels, and one leaf level. The B-tree structure can continue to grow in this way to a maximum of 20 levels.

Branch nodes can point either to other branch nodes below them (for very large indexes of four levels or greater) or to leaf nodes. In Figure 38-31, the branch node points to leaf nodes only. The first item in the left branch node contains the same key value as the largest item in the left-most leaf node and a node pointer to it. The next item contains the largest item in the next leaf node and a node pointer to it, and so on.

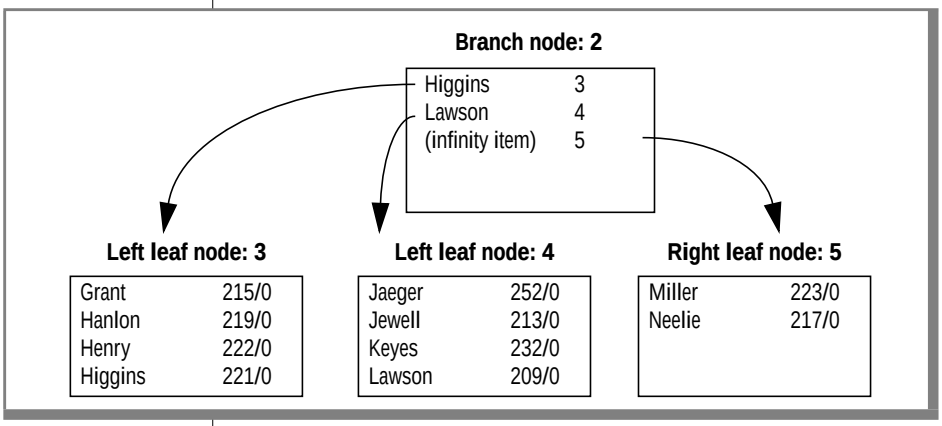


Figure 38-31
Typical Contents of a Branch Node

Duplicate Key Values

Duplicate key values occur when the value of an indexed column is identical for multiple rows. For example, suppose that the third and fourth leaf nodes of a B-tree structure contain the key value **Smith**. Suppose further that this value is duplicated six times, as illustrated in Figure 38-32.

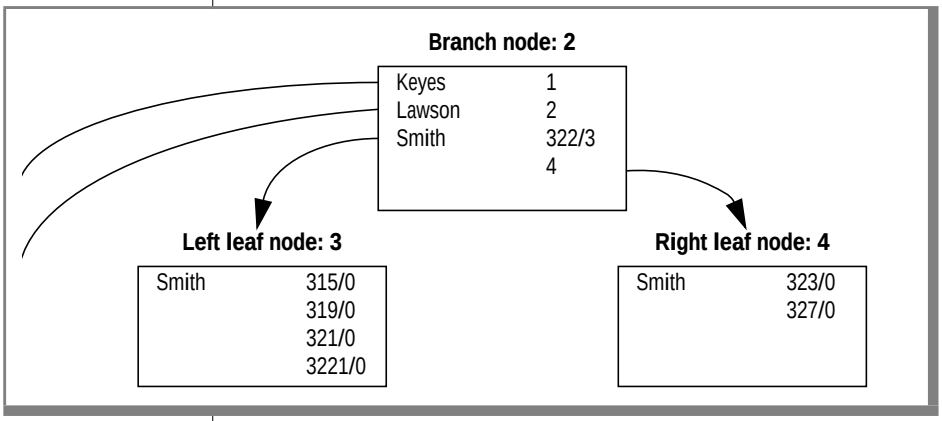


Figure 38-32
Leaf Nodes 3 and 4

The first item on the third leaf page contains the duplicate key value, **Smith**. It also contains the rowid of the first physical row in the table that contains the duplicate key value and a delete flag of 0. To conserve space, the second item does not repeat the key value **Smith** but instead contains just the rowid of the next physical row and a delete flag. This process continues throughout the page; no other key values are on the leaf, only rowid / delete-flag pairs, sorted by rowid.

The first item on the fourth leaf page again contains the duplicated key value and a rowid / delete-flag pair. Subsequent items contain only rowid / delete-flag pairs. Again, the rowids are sorted, and all have a rowid greater than the largest rowid on the third leaf page.

Now consider the branch node. The third item in the branch node contains the same key value *and rowid* as the largest item in the third leaf node and a node pointer to it. The fourth item would contain only a node pointer to the fourth leaf node, thus saving the space of an additional duplicate key value.

Physical Storage Format of Index Pages

An empty node is a 2- or 4-kilobyte area with a page header at the top and a stamp and empty slot-table at the bottom (see Figure 38-33). The page header contains the following information:

- Location of page
- Data used for consistency checking
- Amount and location of free space
- Links to left and right index pages at the same level

The last item, horizontal links to sibling index pages, is unique to index pages. All nodes on the same level are horizontally linked, branch-to-branch or leaf-to-leaf. These links are set to zero for root nodes because no other index pages are at the root-node level.

The horizontal links are implemented as pointers to the next node. These pointers, which are stored in the branch- or leaf-node page header, facilitate sequential index scans. A sequential index scan occurs when the database server traverses a table in index order.

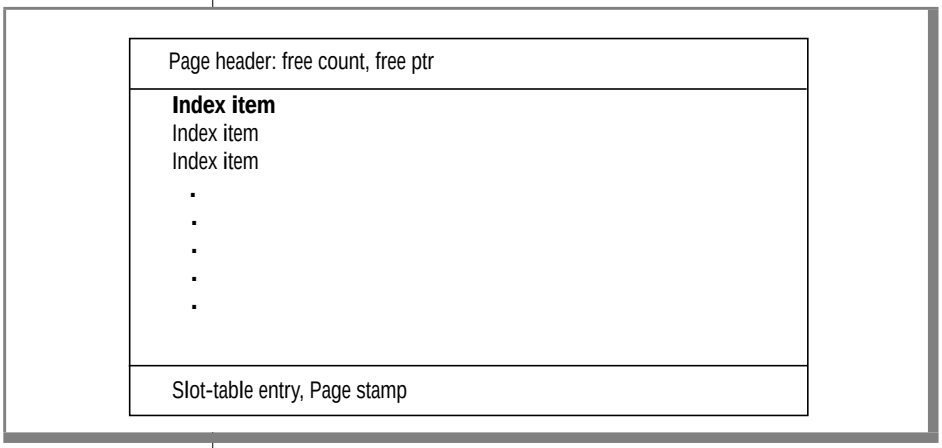


Figure 38-33
Physical Format of Newly Created Index Page

The slot table grows upward from the bottom of the page as the database server adds index items to the page. The slot-table entries point to the physical location of the various items on the page as they are added.

What Happens Physically When You Insert Indexed Data?

When an index item is added to a node, the database server takes the following actions:

1. Sets the length of the item to the size of the key value plus rowid / delete flags
2. Copies the index item to the location indicated by the page free pointer, if contiguous free space exists
3. Allocates a slot structure at page bottom and initializes it with a pointer to the item and length of the item
4. Increments the free pointer by the length of the index item
5. Decrements the free counter by length plus size of slot structure

Key Value Locking

When the database server deletes a row with an index from a table, the delete-flag portion of the index item is marked with a 1, and an exclusive lock is placed on that index item. The database server does not delete the index item from the index until you or your application commit the transaction. After the transaction is committed, a request to delete the item is placed in a list in shared memory called the *btree cleaner list*. This request consists of the tblspace number, the page number of the index page, and the index number for the key to be deleted. The requests in the list are read by the btree cleaner thread at one-minute intervals or if the number of requests in the list exceeds 100; the thread then finds the index page and deletes the index item that is marked as deleted.

You can also use the UPDATE STATISTICS statement to remove deleted items. This statement acts as a backup for the normal method of removing deleted index items. You might want to use this method, for example, when your system fails resulting in outstanding delete requests. For additional information about the UPDATE STATISTICS statement, see the *Informix Guide to SQL: Syntax*.

Are Freed Index Pages Reused?

When the database server physically removes an index item from a node, it checks if the node is a compression candidate. A node becomes a compression candidate when the node contains two or less index items, and the node is not a root node.

When the database server finds a compression candidate, the items contained in the candidate are either merged or shuffled to another node immediately to the right or left.

Merging

The database server merges items if the node to the right or left has space for all the items contained in the compression candidate. When all the items are merged with this sibling, the compression candidate is freed to be reused in the table for any purpose necessary.

Shuffling

If neither of the siblings of a compression candidate has space for all of the items, the database server moves items from the fuller of the two siblings to the compression candidate until they have an approximately equal amount of items. This process is termed *shuffling*.

Controlling How Indexes Are Filled

When you create an index, you can specify how densely or sparsely filled you want the index. The index fill factor is a percentage of each index page that will be filled during the index build. Use the FILLFACTOR option of the CREATE INDEX statement or the FILLFACTOR configuration parameter to set the fill factor. This option is particularly useful for indexes that you do not expect to grow after they are built. For additional information about the FILLFACTOR option of the CREATE INDEX statement, see the *Informix Guide to SQL: Syntax*.

Calculating the Length of Index Items

Figure 38-34 illustrates the physical-storage format of index items. As shown in this illustration, index items typically contain a key value and one or more rowid / delete-flag pairs. For data types other than VARCHAR, the length of an index item is calculated by adding the length of the key value plus 5 bytes for each rowid / delete flag associated with the key value.

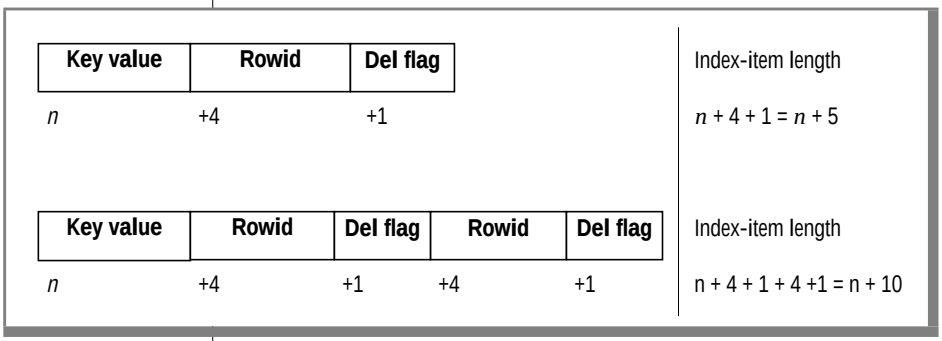


Figure 38-34
Index-Item Formats

The key values in an index are typically of fixed length. If an index holds the value of one or more columns of the VARCHAR data type, the length of the key value is at least the sum of the length-plus-one of each VARCHAR value in the key. A byte precedes the VARCHAR data in both the row and the key value in the index. This byte contains the actual length of the VARCHAR column. Because the maximum length of a key value is 390 bytes, the combined size of VARCHAR columns that make up a key must be less than 390, minus an additional byte for each VARCHAR column. For example, the key length of the index that the database server builds for the following statements equals 390, or ((1+255) + (1+133)):

```
CREATE TABLE T1 (c1 varchar(255, 10), c2 varchar(133, 10));
CREATE INDEX I1 on T1(c1, c2);
```

Figure 38-34 illustrates the structure of an index item.

Blobspace Structure and Storage

This section explains the structures and storage techniques that the database server uses to store TEXT and BYTE data in a blobspace.

Structure of a Blobspace

When you create a blobspace, you can specify the effective size of the data pages, which are called blobpages. The blobpage size for the blobspace is specified when the blobspace is created. Blobpage size must be a multiple of page size. (See "Determining Database Server Page Size" on page 14-17.) All blobpages within a blobspace are the same size, but the size of the blobpage can vary between blobspaces. Blobpage size can be greater than the page size because data stored in a blobspace is never written to the page-sized buffers in shared memory.

The advantage of customizing the blobpage size is storage efficiency. Within a blobspace, TEXT and BYTE data types are stored in one or more blobpages, but objects do not share blobpages. Storage is most efficient when the TEXT or BYTE data is equal to, or slightly smaller than, the blobpage size.

The blobspace free-map pages and bit-map pages are the size specified as a database-server page, which enables them to be read into shared memory and to be logged.

When the blobspace is first created, it contains the following structures:

- Blobspace free-map pages
- The blobspace bit map that tracks the free-map pages
- Unused blobpages

Figure 38-35 illustrates the chunk structure of a blobspace as it appears immediately after the blobspace is created.

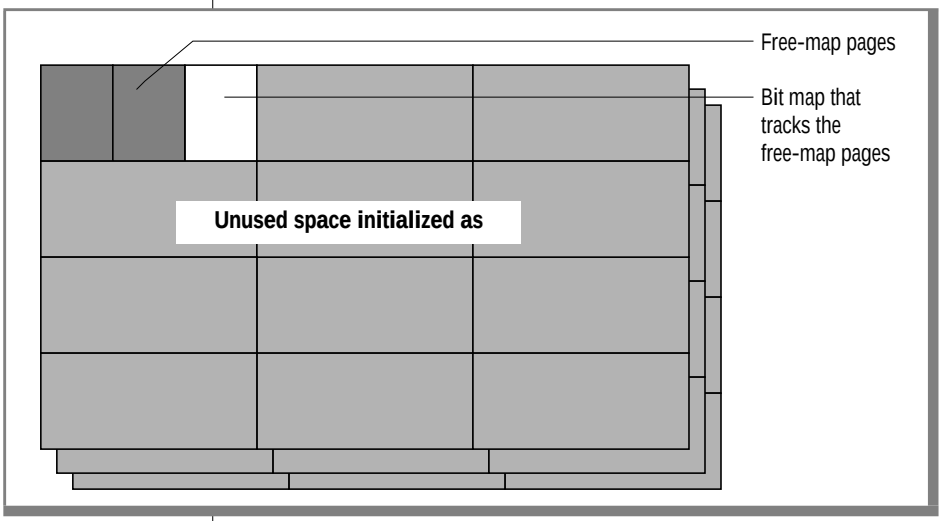


Figure 38-35
Structure of a Chunk in a Blobspace

TEXT and BYTE Data Storage and the Descriptor

Data rows that include TEXT or BYTE data do not include the data in the row itself. Instead, the data row contains a 56-byte descriptor that includes a forward pointer (rowid) to the location where the first segment data is stored. The descriptor can point to a page (if the data is stored in a dbspace), a blobpage (if the data is stored in a blobspace), or an optical platter (if you are using the Optical Subsystem).

Figure 38-36 illustrates the structure of the 56-byte descriptor.

Figure 38-36
Structure of the Descriptor for TEXT or BYTE data

```
typedef struct tblob
    {
    short   tb_fd;      /* TEXT or BYTE data descriptor (must be first) */
    short   tb_coloff;  /* Column offset in row */
    long    tb_tblspace;/* TEXT or BYTE table space */
    long    tb_end;     /* ending byte: 0 for end of TEXT or BYTE data */
    long    tb_size;    /* Size of data */
    long    tb_addr;    /* Starting Sector or Blobpage */
    long    tb_family;  /* Family Number (optical support) */
    long    tb_volume;  /* Family Volume */
    short   tb_medium;  /* Medium - one if optical */
    short   tb_bstamp;  /* first Blobpage stamp */
    short   tb_sockid;  /* socket id of remote blob */
    short   tb_flags;   /* flags */
    long    tb_sysid;   /* optical system identifier */
    long    tb_reserved2;/* reserved for the future */
    long    tb_reserved3;/* reserved for the future */
    long    tb_reserved4;/* reserved for the future */
    } tblob_t;
```

When Are TEXT or BYTE Objects Created?

When a row that contains TEXT or BYTE data is to be inserted, the TEXT or BYTE objects are created first. After the objects are written to disk (or optical medium), the row is updated with the descriptor and inserted.

Are TEXT and BYTE Objects Modified?

TEXT and BYTE objects are never modified. TEXT and BYTE objects can only be inserted or deleted. Deleting an object means that the database server frees the space consumed by the deleted blob for reuse.

When TEXT or BYTE data is updated, a new TEXT or BYTE object is created, and the data row is updated with the new blob descriptor. The old image of the row contains the descriptor that points to the obsolete value for the TEXT or BYTE object. The space consumed by the obsolete TEXT or BYTE object is freed for reuse after the update is committed. TEXT or BYTE objects are automatically deleted if the rows containing their blob descriptors are deleted. (Blobpages that stored a deleted TEXT or BYTE object are not available for reuse until the logical log that contains the original INSERT record for the deleted TEXT or BYTE object is backed up. For more information, see "Backing Up Logical-Log Files to Free Blobpages" on page 18-22)

What Limits the Size of TEXT and BYTE Objects?

The largest TEXT or BYTE object that the blob descriptor can accommodate is $(2^{31} - 1)$, or about 2 gigabytes. This limit is imposed by the 4-byte integer that defines the size of the TEXT or BYTE object in the blob descriptor.

Structure of a Dbspace Blobpage

TEXT or BYTE data that is stored in the dbspace is stored in a blobpage. The structure of a dbspace blobpage is similar to the structure of a dbspace data page. The only difference is an extra 12 bytes that can be stored along with the TEXT or BYTE data in the data area.

TEXT and BYTE objects can share dbspace blobpages if more than one TEXT or BYTE object can fit on a single page, or if more than one trailing portion of a TEXT or BYTE object can fit on a single page.

For a general discussion of how to estimate the number of dbspace blobpages needed for a specific table, see your *Performance Guide*.

Each segment of TEXT or BYTE data stored in a dbspace page might be preceded by up to 12 bytes of information that do not appear on any other dbspace page. These extra bytes contain up to three pieces of information:

- A 4-byte blob time stamp for this blob segment (required)
- A 4-byte forward pointer (rowid) to the next portion of the blob segment, if one exists (optional)
- A 4-byte blob time stamp stored with the forward pointer to the next portion of the blob segment (required if a forward pointer exists)

Figure 38-37 illustrates TEXT or BYTE data storage in a dbspace.

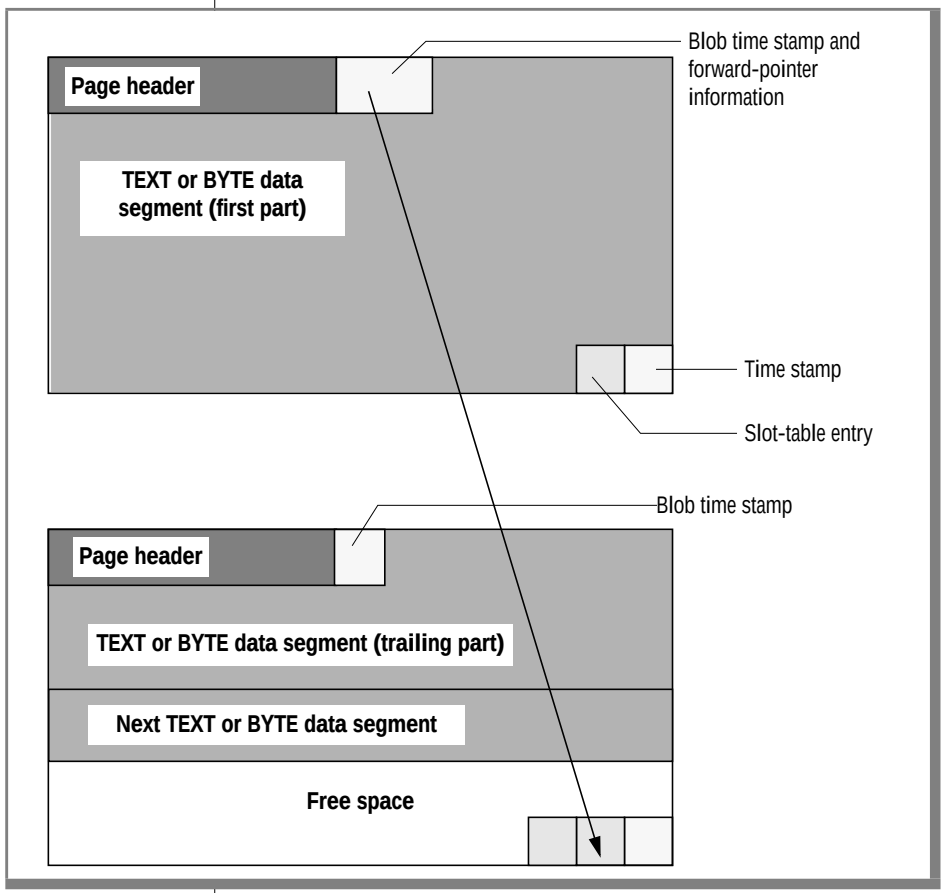


Figure 38-37
Dbspace Blobpage

Blobspace Page Types

Every blobspace chunk contains three types of pages:

- A blobspace free-map page
- A bit-map page
- Blobpages

What Is the Blobspace Free-Map Page?

The blobspace free-map page identifies unused blobpages so that the database server can allocate them as part of blob creation. When a blobpage is allocated, the free-map entry for that page is updated. All entries for a single TEXT or BYTE object are linked.

A blobspace free-map page is the size of one database server page. Each entry on a free-map page is 8 bytes, stored as two 32-bit words, as follows:

- The first bit in the first word specifies whether the blobpage is free or used.
- The next 31 bits in the first word identify the logical-log file that was current when this blobpage was written. (This information is needed for logging TEXT or BYTE data.)
- The second word contains the tblspace number associated with the TEXT or BYTE object stored on this page.

The number of entries that can fit on a free-map page depends on the page size of your computer. The number of free-map pages in a blobspace chunk depends on the number of blobpages in the chunk.

What Is the Blobspace Bit-Map Page?

The blobspace bit-map page tracks the fullness and number of blobspace free-map pages in the chunk. Each blobspace bit-map page is capable of tracking a quantity of free-map pages that represent more than 4,000,000 blobpages. Each blobspace bit-map page is the size of one page.

What Is the Blobpage?

The blobpage contains the TEXT or BYTE data. Blobpage size is specified by the database server administrator who creates the blobspace. Blobpage size is specified as a multiple of the page size.

Structure of a Blobspace Blobpage

The storage strategy used to store TEXT or BYTE objects in a blobspace differs from the dbspace storage strategy. The database server does not combine whole TEXT or BYTE objects or portions of a TEXT or BYTE object on a single blobspace blobpage. For example, if blobspace blobpages are 24 kilobytes each, a TEXT or BYTE object that is 26 kilobytes is stored on two 24-kilobyte pages. The extra 22 kilobytes of space remains unused.

Blobpage Structure

The structure of a blobpage includes a blobpage header, the TEXT or BYTE data, and a page-ending time stamp. The blobpage header includes, among other information, the page-header time stamp and the blob time stamp associated with the forward pointer in the data row. If a TEXT or BYTE object is stored on more than one blobpage, a forward pointer to the next blobpage and another blob time stamp are also included in the blobpage header.

Figure 38-38 illustrates the structure of a blobpage.

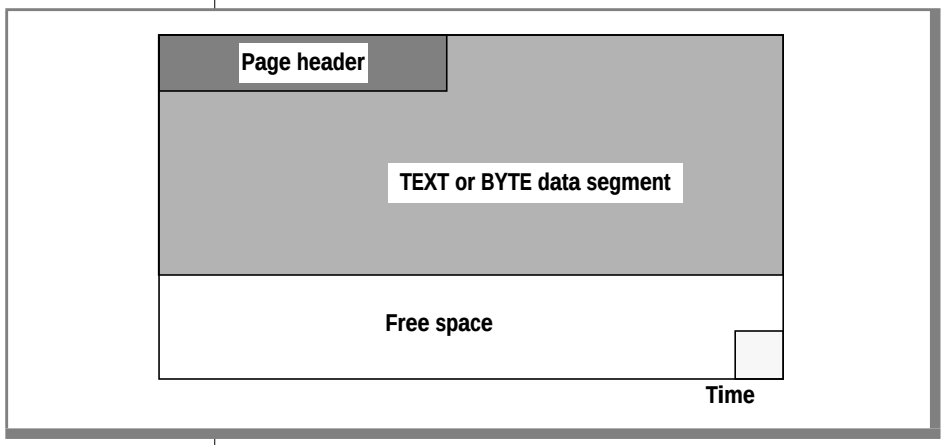


Figure 38-38
Blobspace Blobpage

What Is in the Blobpage Header?

The blobpage header includes the following information:

- The physical address of the blobpage
- A page-header time stamp that indicates the last time this blobpage was modified
- A forward pointer to the blobpage that holds the next segment of TEXT or BYTE data and an associated blob time stamp, if a next segment exists

  Otherwise, only the current page number appears, indicating this is the last page.
- A blob time stamp that describes the last time this page was allocated (when TEXT or BYTE data was written to the page)
- The size of this blobpage
- A percentage of blobpage fullness
- A unique identifier that is written when a blobpage is written to tape (used only during the data-restore procedure)

Figure 38-39 on page 38-67 illustrates the different locations of the two pairs of time stamps that appear on the blobspace blobpage

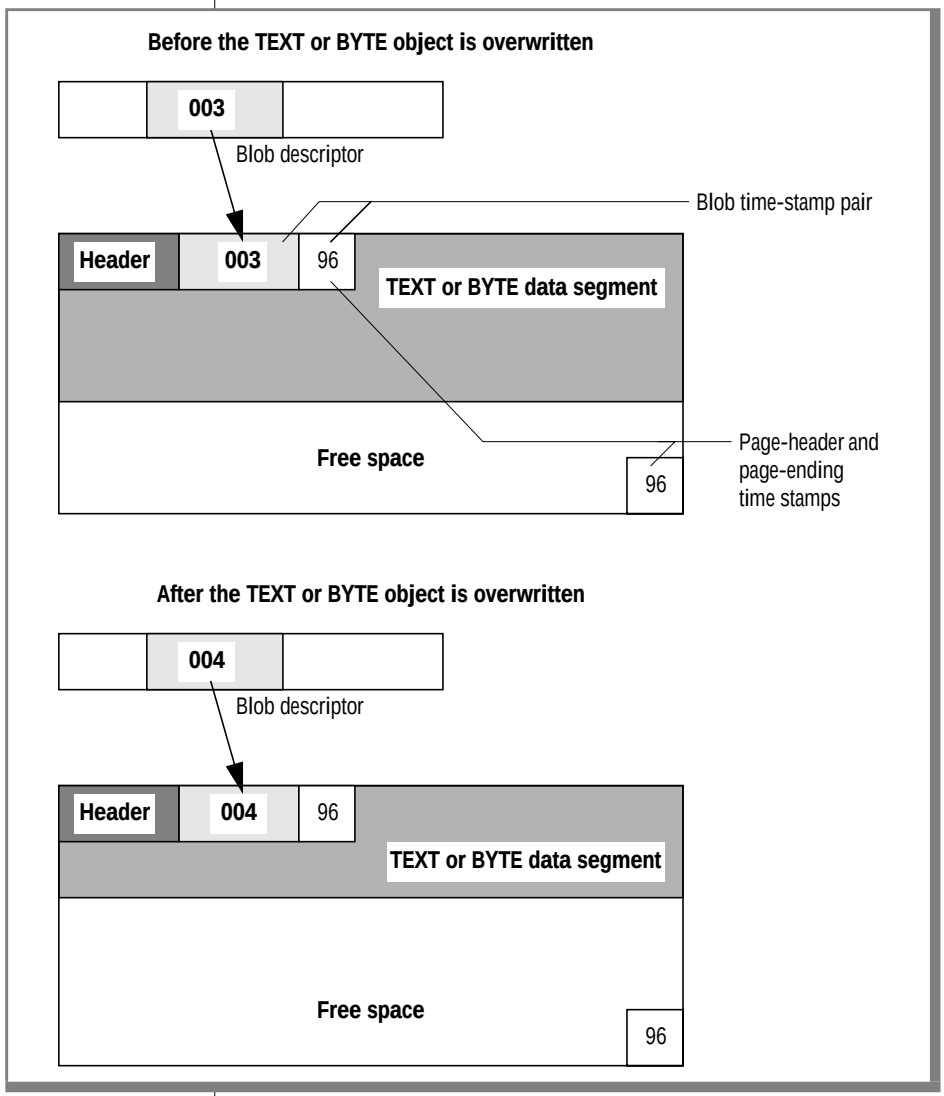


Figure 38-39
Blobpage Time Stamps

The blob time stamps track the point at which a blobpage is allocated. Page-header and page-ending time stamps validate page consistency and confirm that the page write was successful.

Database and Table Creation: What Happens on Disk

This section explains how the database server stores data related to the creation of a database or table and allocates the disk structures that are necessary to store your data.

Creating a Database

After the root dbspace exists, users can create a database. The paragraphs that follow describe the major events that occur on disk when the database server adds a new database.

Disk-Space Allocation for System Catalog Tables

The database server searches the chunk free-list pages (see "Structure of the Chunk Free-List Page" on page 38-18) in the dbspace, looking for free space in which to create the system catalog tables. For each system catalog table, in turn, the database server the database server allocates eight contiguous pages, the size of the initial extent of each system catalog table. The tables are created individually and do not necessarily reside next to each other in the dbspace. They can be located in different chunks. As adequate space is found for the initial extent of each table, the pages are allocated, and the associated chunk free-list page is updated.

System Catalog Tables Are Tracked

The database server tracks newly created databases in the database tblspace, which resides in the root dbspace. An entry describing the database is added to the database tblspace (see "Structure of the Database Tblspace" on page 38-24) in the root dbspace. For each system catalog table, the database server adds a one-page entry to the tblspace tblspace (see "Structure of the Tblspace Tblspace" on page 38-19) in the dbspace where the database was built. Figure 38-40 illustrates the relationship between the database tblspace entry and the location of the **systables** table for the database.

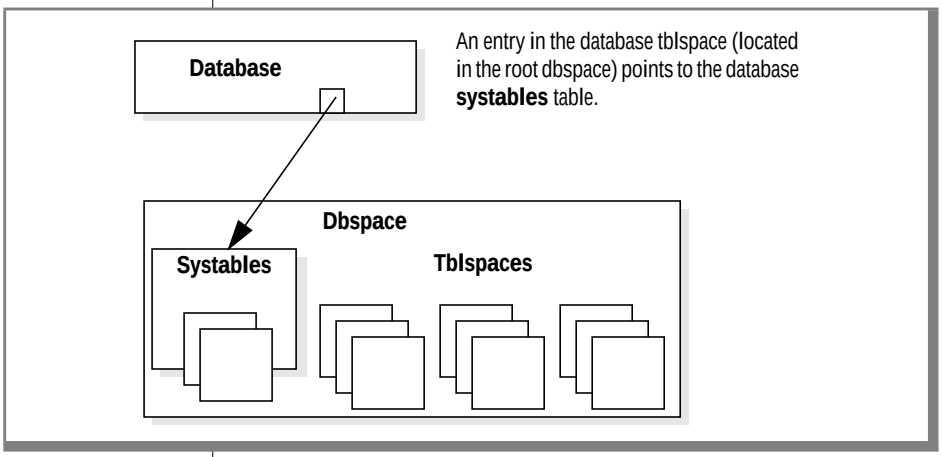


Figure 38-40
New Databases

For instructions on how to list your databases after you create them, see "Monitoring Databases" on page 29-43.

Creating a Table

After the root dbspace exists, and a database has been created, users with the necessary SQL privileges can create a database table. When users create a table, the database server allocates disk space for the table in units called extents (see "What Is an Extent?" on page 13-13). The paragraphs that follow describe the major events that occur when the database server creates a table and allocates the initial extent of disk space.

Disk-Space Allocation

The database server searches the chunk free-list pages (see "Structure of the Chunk Free-List Page" on page 38-18) in the dbspace for contiguous free space equal to the initial extent size for the table. When adequate space is found, the pages are allocated, and the associated chunk free-list page is updated. If space for the extent cannot be found, an error is returned. (Because an extent is, by definition, contiguous disk space, extents cannot span two chunks.)

Entry Is Added to Tblspace Tblspace

The database server adds a one-page entry for this table to the tblspace tblspace in this dbspace. The tblspace number (see "What Is the Tblspace Number?" on page 38-21) assigned to this table is derived from the logical page number in the tblspace tblspace where the table is described.

The tblspace number indicates the dbspace where the tblspace is located. Tblspace extents can be located in any of the dbspace chunks. Execute **oncheck -pe** for a listing of the dbspace layout by chunk if you must know exactly where the tblspace extents are located.

Entries Are Added to the System Catalog Tables

The table itself is fully described in entries stored in the system catalog tables for the database. Each table is assigned a table identification number or *tabid*. The tabid value of the first user-defined table in a database is always 100. For a complete discussion of the system catalog, see the *Informix Guide to SQL: Reference*.

Figure 38-41 illustrates the pointers within the disk data structures that track and monitor the disk space allocated to a table.

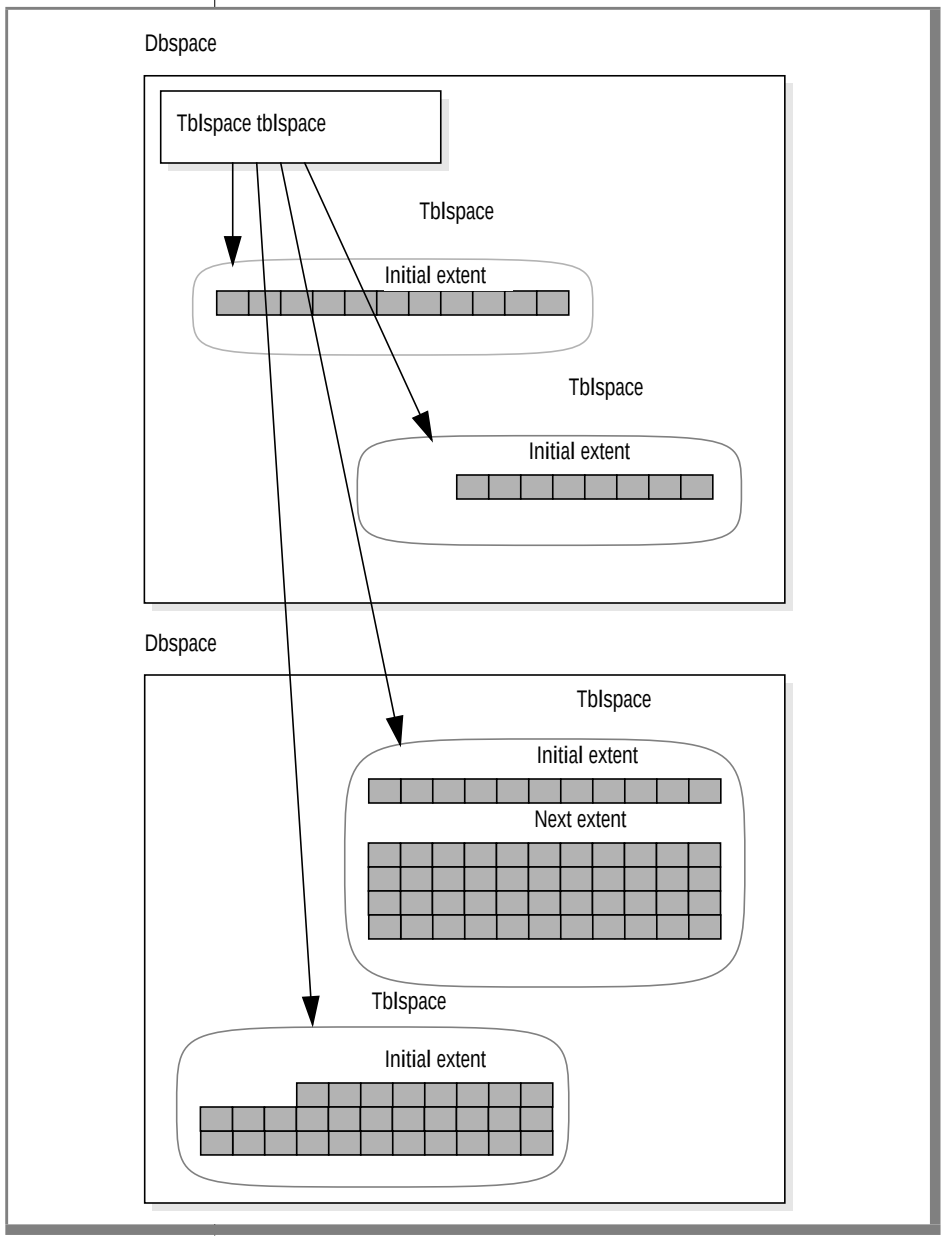


Figure 38-41
New Tables

A table can be located in a dbspace that is different than the dbspace that contains the database. The tblspace itself is the sum of allocated extents, not a single, contiguous allocation of space. The database server tracks tblspaces independently of the database.

What Happens on Disk When a Temporary Table Is Created?

The tasks involved in creating temporary tables are similar to the tasks that the database server performs when it adds a new permanent table. The key difference is that temporary tables do not receive an entry in the system catalog for the database. For more information on temporary tables, see "What Is a Temporary Table?" on page 13-24.

Appendix

A

Files That the Database Server Uses

This appendix provides brief summaries of the files that you use when you configure and use Dynamic Server. The filenames are shown in a UNIX format. For Windows NT, the file names are the same with the following format:

```
maindirectory\subdirectory\subdirectory
```

This appendix also includes descriptions of some of the files and one directory that the database server creates and uses internally. For many of these files, your only responsibility is to recognize that they are legitimate.

Figure A-1 lists the database server files and the directories that contain them. In the table in Figure A-1 on page A-2 and throughout the chapter, the directory examples are shown using UNIX pathname syntax. The Windows NT directories use the same filenames. The following table provides examples of the difference between UNIX pathname syntax and Windows NT pathname syntax.

UNIX Syntax	Windows NT Syntax
/usr/lib	**\usr\lib**
$INFORMIXDIR/etc	**%INFORMIXDIR%\etc**

Figure A-1
Files that the Database Server Uses

Filename	Directory	Purpose	Created
af.*xxx*	specified by DUMPDIR configuration parameter	assertion-failure information	by the database server
buildsmi.*xxx*	**/tmp**	error messages about SMI database	by the database server
config.arc	**$INFORMIXDIR/etc**	configuration for archiving	during installation; modified by user **informix** ♦
core	directory from which the database server was invoked	core dump	by the database server ♦
gcore	specified by DUMPDIR configuration parameter	assertion failure information	by the database server ♦
i*lllsrra.xx*	On UNIX: **lib** and **/usr/lib** On Windows NT: **lib**	shared libraries for the database server and some utilities	by install procedure
.informix	user's home directory	set personal environment variables	by the user
informix.rc	**$INFORMIXDIR/etc**	set default environment variables for all users	by user **informix**
Directory: **INFORMIXTMP**	**INFORMIXTMP**	create and maintain local files	by the database server

(1 of 3)

Filename	Directory	Purpose	Created
.inf.*servicename*	**/INFORMIXTMP**	connection information	by the database server
.infos.*servername*	**$INFORMIXDIR/etc**	connection information	by the database server
message log; filename specified by MSGPATH configuration parameter	specified by MSGPATH	error messages and status information	by the database server
ONCONFIG file; filename specified by ONCONFIG environment variable	**$INFORMIXDIR/etc**	configuration information	by user **informix**
onconfig	**$INFORMIXDIR/etc**	default ONCONFIG file (optional)	by user **informix**
onconfig.std	**$INFORMIXDIR/etc**	template for ONCONFIG file	during installation
oncfg_*servername.servernum*	**$INFORMIXDIR/etc**	information for full-system restores	by the database server
oper_dflt.arc	**$INFORMIXDIR/etc**	defaults for archiving	during installation; modified by user **informix** ♦
***servicename*.exp**	**/INFORMIXTMP**	connection information	by the database server
***servicename*.str**	**/INFORMIXTMP**	connection information	by the database server

UNIX

(2 of 3)

UNIX

Filename	Directory	Purpose	Created
shmem.*xxx*	specified by DUMPDIR configuration parameter	assertion-failure information	by the database server
sqlhosts	**$INFORMIXDIR/etc**	connection information	during installation; modified by user **informix** ♦
VP.*servername.nnx*	**/INFORMIXTMP**	connection information	by the database server

(3 of 3)

Descriptions of Files

This section gives short descriptions of the files listed in Figure A-1.

af.xxx

The database server writes information about an assertion failure into the **af.*xxx*** file. The file is stored in the directory specified by the DUMPDIR configuration parameter. For more information, see "Monitor for Data Inconsistency" on page 27-7.

buildsmi.xxx

If the database server cannot build the **sysmaster** database, it places a message in the message log that refers you to the **buildsmi.*xxx*** file. This file provides information about why the build failed. For information about the **sysmaster** database, see Chapter 34, "The sysmaster Database."

UNIX

core

The **core** file contains a core dump caused by an assertion failure. The database server writes this file into the directory from which the database server was invoked. For more information, see "Monitor for Data Inconsistency" on page 27-7.

UNIX

gcore.xxx

The database server writes information about an assertion failure into the **gcore.*xxx*** file. The file is stored in the directory specified by the DUMPDIR configuration parameter. For more information, see "Monitor for Data Inconsistency" on page 27-7.

illlsrra.xx

The ***illlsrra.xx*** files are shared libraries that the database server and some database server utilities use. The shared libraries, if supported on your platform, are installed in **lib**. Additionally on UNIX, symbolic links to them are automatically created in **/usr/lib** when the products are installed on your computer.

The naming convention of the Informix shared library filename is:

```
illlsrra.xx
```

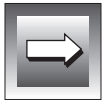
lll library class (for example, "asf" or "smd")

s library subclass (d=DSA; s=standard)

rr major release number (for example, "07" or "08")

a library version ID (for example, "a" or "b")

xx shared-library filename extension (for example, "so")



Important: *The symbolic links to the shared libraries in **/usr/lib** are automatically created by the product installation procedures on UNIX. However, if your **$INFORMIXDIR** is not installed using the standard installation method (for example. your **$INFORMIXDIR** is NFS-mounted from another computer), you or your system administrator may need to manually create the symbolic links of the shared libraries in **/usr/lib** on your computer.*

.informix

The **.informix** file (**~/.informix** on UNIX) is the *private-environment file*. Users can create this file and store it in their home directory. Chapter 4 of the *Informix Guide to SQL: Reference* discusses the environment-configuration files.

informix.rc

The **informix.rc** file is the *environment-configuration file*. You can use it to set environment variables for all users of Informix products. Chapter 4 of the *Informix Guide to SQL: Reference* discusses the environment-configuration files.

INFORMIXTMP

The **INFORMIXTMP** directory is an *internal database server directory*. During initialization, the database server creates this directory (if it does not exist yet) for storing internal files that must be local and relatively safe from deletion.These internal files cannot be saved in **INFORMIXDIR** or **tmp** because **INFORMIXDIR** is not necessarily local, and files in **tmp** can easily be deleted by accident.

.inf.servicename

The database server creates **.inf.*servicename*** if any DBSERVERNAME or DBSERVERALIASES uses a shared-memory connection. The database server removes the file when you take the database server off-line. The database server derives the name of this file from the servicename field of the **sqlhosts** file. The database server keeps information about client/server connections in this file.

You do not use **.inf.*servicename*** directly. You only need to recognize that it is a legitimate file when it appears in the **INFORMIXTMP** directory.

If this file is accidentally deleted, you must restart the database server.

.infos.servername

The database server creates **.infos.*servername*** when you initialize shared memory and removes the file when you take the database server off-line. The database server derives the name of this file from the DBSERVERNAME parameter in the ONCONFIG file. The database server uses this file to attach to utilities such as **onstat**.

You do not use **.infos.*servername*** directly. You only need to recognize that the file is a legitimate file when it appears in your **INFORMIXDIR/etc** directory.

Message Log

The database server writes status and error information into the message-log file. Use the MSGPATH configuration parameter to specify the filename and location of the message log. For more information, see "MSGPATH" on page 33-55.

oncatlgr.out.pidnum

The **oncatlgr.out.*pidnum*** file is created when the **oncatlgr** utility is started using the **start_oncatlgr** script. The file contains output messages from **oncatlgr**. For information about **oncatlgr**, see the *Archive and Backup Guide*.

ONCONFIG File

The **ONCONFIG** file is the *current configuration file*. The database server uses the ONCONFIG file during initialization. To maintain compatibility with earlier versions, the database server also recognizes the **TBCONFIG** environment variable. However, Informix recommends that you use **ONCONFIG**.

If you start the database server with **oninit** and do not explicitly set the **ONCONFIG** environment variable, the database server uses the configuration file specified by the **TBCONFIG** environment variable. If neither **ONCONFIG** nor **TBCONFIG** is set, and **onconfig** does not exist, the database server returns the following error message:

```
WARNING: Cannot access configuration file $INFORMIXDIR/etc/$ONCONFIG.
```

For more information, see Chapter 3, "Configuring the Database Server."

onconfig

The **onconfig** file is an optional file in the **etc** directory. It contains configuration information. You can create the **onconfig** file by copying **onconfig.std** or one of your customized configuration files.

onconfig.std

The **onconfig.std** file is the *configuration-file template*. It contains default values for the configuration parameters and serves as the template for all ONCONFIG files that you create. To use the template, copy it into another file, and modify the values. Do not modify **onconfig.std**.

For more information, see "A Sample onconfig.std File" on page A-10, "Configuring the Database Server" on page 3-7, and Chapter 33, "Configuration Parameters".

oncfg_servername.servernum

The database server creates **oncfg_*servername.servernum*** when you initialize disk space (**oninit -i**). The database server updates **oncfg_*servername.servernum*** every time you add or delete a dbspace, a logical-log file, or a chunk. The database server uses **oncfg_*servername.servernum*** when it salvages logical-log files during a full-system restore. The database server derives the name of this file from the values of the DBSERVERNAME and SERVERNUM parameters in the ONCONFIG file.

You do not use **oncfg_*servername.servernum*** directly. You only need to recognize that these files are legitimate files when they appear in your **etc** directory. For more information, see "Create the oncfg_servername.servernum File" on page 8-10.

UNIX

oper_deflt.arc

The **oper_deflt.arc** file contains default command qualifier values used by ON-Archive. The *Archive and Backup Guide* describes the **oper_dflt.arc** file.

Users can set the **ARC_DEFAULT** environment variable to point to a file that contains personal defaults that differ from the defaults in **oper_deflt.arc**.

servicename.exp

On some platforms, the database server creates ***servicename*.exp** if a DBSERVERNAME or DBSERVERALIASES uses a stream-pipe connection. The database server uses ***servicename*.exp** to handle expedited data, which occurs when the client program issues sqlbreak( ) or the user enters CTRL-C. The database server removes ***servicename*.exp** when you take the database server off-line. The database server derives the name of this file from the servicename field of the **sqlhosts** file.

You do not use the ***servicename*.exp** file directly. You only need to recognize that it is a legitimate file when it appears in the **INFORMIXTMP** directory.

servicename.str

The database server creates ***servicename*.str** if a DBSERVERNAME or DBSERVERALIASES uses a stream-pipe connection. The database server removes ***servicename*.str** when you take the database server off-line. The database server derives the name of this file from the servicename field of the **sqlhosts** file. The database server keeps information about client/server connections in ***servicename*.str**.

You do not use ***servicename*.str** directly. You only need to recognize that it is a legitimate file when it appears in the **INFORMIXTMP** directory.

If this file is accidentally deleted, you must restart the database server.

shmem.xxx

The database server writes information about an assertion failure into the **shmem.*xxx*** file. The file is stored in the directory specified by the DUMPDIR configuration parameter. For more information, see "Monitor for Data Inconsistency" on page 27-7.

UNIX

sqlhosts

The **sqlhosts** file is the *connectivity file*. It contains information that lets an Informix client application connect to an Informix database server. For more information, see Chapter 4, "Client/Server Communications."

Tip: *On Windows NT, the connectivity information is in a key in the Windows registry called* ***HKEY_LOCAL_MACHINE\SOFTWARE\Informix\SQLHOSTS****.*

VP.servername.nnx

The database server creates **VP.*servername.nnx***, if needed, when you initialize shared memory. The database server derives the name of this file from the DBSERVERNAME or DBSERVERALIASES in the ONCONFIG file, the VP number (*nn*), and an internal identifier (*x*). The database server keeps information about client/server connections in **VP.*servername.nnx***.

You do not use the file directly. You only need to recognize that it is a legitimate file.

If this file is accidentally deleted, you must restart the database server.

A Sample onconfig.std File

The following file is a sample copy of the **onconfig.std** file. Some of the values might be different from the ones in your **onconfig.std** file because some values are platform dependent.

```
#**************************************************************************
#
#       INFORMIX SOFTWARE, INC.
#
#  Title:onconfig.ids
#  Description: Informix Dynamic Server Configuration Parameters
#
#**************************************************************************

# Root Dbspace Configuration

ROOTNAMErootdbs      # Root dbspace name
ROOTPATH/dev/dynsrv_root  # Path for device containing root dbspace
ROOTOFFSET0      # Offset of root dbspace into device (Kbytes)
ROOTSIZE50000      # Size of root dbspace (Kbytes)

# Disk Mirroring Configuration Parameters

MIRROR0      # Mirroring flag (Yes => 1, No => 0)
MIRRORPATH      # Path for device containing mirrored root
MIRROROFFSET0      # Offset into mirrored device (Kbytes)

# Physical Log Configuration

PHYSDBSrootdbs       # Location (dbspace) of physical log
PHYSFILE5000      # Physical log file size (Kbytes)

# Logical Log Configuration

LOGFILES10      # Number of logical log files
LOGSIZE1000      # Logical log size (Kbytes)

# Configuration Manager

CONFIGSIZE STANDARD # Amt of rootdbs space to reserver for CM data
           # Only applies to CM coservers

# Diagnostics

MSGPATH/usr/informix/online.log   # System message log file path
CONSOLE/dev/console      # System console message path
ALARMPROGRAM      # Alarm program path

# System Configuration

SERVERNUM102      # Unique id corresponding to a db server instance
DBSERVERNAMExyz32t_prod       # Name of default database server
DBSERVERALIASES      # List of alternate dbservernames
NETTYPEipcshm,1,20,CPU  # Override sqlhosts nettype parameters
DEADLOCK_TIMEOUT   60      # Max time to wait of lock in distributed env.
RESIDENT0      # Forced residency flag (Yes = 1, No = 0)

MULTIPROCESSOR 0      # 0 for single-processor,
             # 1 for multi-processor
```

```
NUMCPUVPS1      # Number of user (cpu) vps
SINGLE_CPU_VP 0       # If non-zero, limit number of cpu vps to one

NOAGE0       # Process agingy
AFF_SPROC0      # Affinity start processor
AFF_NPROCS0      # Affinity number of processors

# Shared Memory Parameters

LOCKS5000      # Maximum number of locks
BUFFERS2000      # Maximum number of shared buffers
NUMAIOVPS      # Number of asynchronous I/O VPs
NUMFIFOVPS2 # Number of FIFO VPs
PHYSBUFF32      # Physical log buffer size (Kbytes)
LOGBUFF32      # Logical log buffer size (Kbytes)
LOGSMAX100      # Maximum number of logical log files
CLEANERS 1      # Number of buffer cleaner processes
SHMBASE 0x30000000L      # Shared memory base address
SHMVIRTSIZE8000      # Initial virtual shared memory segment size
SHMADD 8192      # Size of new shared memory segments (Kbytes)
SHMTOTAL 0      # Total shared memory (Kbytes). 0=>unlimited
CKPTINTVL 300      # Check point interval (in seconds)
LRUS8        # Number of LRU queues
LRU_MAX_DIRTY60      # LRU percent dirty begin cleaning limit
LRU_MIN_DIRTY50      # LRU percent dirty end cleaning limit
LTXHWM50      # Long transaction high water mark percentage
LTXEHWM60      # Long transaction high water mark
TXTIMEOUT300      # Transaction timeout (in seconds)
STACKSIZE32      # Stack size (Kbytes)

# System Page Size

# BUFFSIZE - Dynamic Server no longer supports this configuration parameter.
#           To determine the page size used by the Dynamic Server on your platform
#            see the last line of output from the command, 'onstat -b'.

# Recovery Variables
# OFF_RECVRY_THREADS:
# Number of parallel worker threads during fast recovery or an offline restore.
# ON_RECVRY_THREADS:
# Number of parallel worker threads during an online restore.

OFF_RECVRY_THREADS   10      # Default number of offline worker threads
ON_RECVRY_THREADS    1      # Default number of online worker threads

# Data Replication Variables
# DRAUTO: 0 manual, 1 retain type, 2 reverse type
DRAUTO0 # DR automatic switchover
DRINTERVAL30      # DR max time between DR buffer flushes (secs)
DRTIMEOUT30      # DR network timeout (in sec)
DRLOSTFOUND/usr/informix/etc/dr.lostfound   # DR lost+found file path

# Read Ahead Variables
RA_PAGES      # Number of pages to attempt to read ahead
```

```
RA_THRESHOLD      # Number of pages left before next group

# DBSPACETEMP:
# This is the list of dbspaces
# that the SQL Engine will use to create temp tables etc.
# The list can be specified using various preset "sets" that are
# ALL - all dbspaces can be used for temp files
# TEMP - use only temporary dbspaces for temp files
# NOTCRITICAL - use only non-critical dbspaces for temp files
#   Critical dbspaces are the rootdbspaces and dbspaces containing
#   log files.
# The list an also be specified as an explicit list of dbspaces/dbslices
# that exist when the Dynamic Server is brought online. If not specified,
# or if all dbspaces specified are invalid, various ad hoq queries will create
# temporary files in /tmp instead. If a dbspaces in this list is
# added to the system at a later point in time, it is automatically added
# to the list of dbspaces that can be used for temp purposes.


DBSPACETEMPNOTCRITICAL# Default temp dbspaces

# DUMP*:
# The following parameters control the type of diagnostics information which
# is preserved when an unanticipated error condition (assertion failure) occurs
# during Dynamic Server operations.
# For DUMPSHMEM, DUMPGCORE and DUMPCORE 1 means Yes, 0 means No.

DUMPDIR/tmp      # Preserve diagnostics in this directory
DUMPSHMEM1      # Dump a copy of shared memory
DUMPGCORE0      # Dump a core image using 'gcore'
DUMPCORE0 # Dump a core image (Warning: this aborts Dynamic)
        # Server
DUMPCNT1      # Number of shared memory or gcore dumps for
              # a single user's session

# ADT*
# The following parameters control the type and level of secure auditing
# present in the Dynamic Server system. By default, ADTMODE is 0 and auditing
# is disabled

ADTMODE0# Auditing mode
ADTPATH/tmp# Directory where audit trails will be written
        # by Dynamic Server
ADTSIZE50000# Maximum size of any single audit trail file
ADTERR0# 0 ==> retry railed audit writes;
        # 1 ==> log failure
# Index Fill Factor

FILLFACTOR90      # Fill factor for building indexes

# Method for Dynamic Server to use when determining current time

USEOSTIME0      # 0: use internal time (fast),
              # 1: get time from OS (slow)
```

```
# Parallel Database Queries (pdq)
PDQPRIORITY0 # Degree of parallism: 0 => OFF, 1 => LOW,
        # -2 => HIGH, max of 100.
MAX_PDQPRIORITY    100      # Maximum allowed pdqpriority
DS_MAX_QUERIES       # Maximum number of decision support queries
DS_TOTAL_MEMORY       # Decision support memory (Kbytes)
DS_MAX_SCANS    1048576      # Maximum number of decision support scans
DATASKIP       # List of dbspaces to skip

# Backup/Restore variables

BAR_ACT_LOG      /tmp/bar_act.log  # File for errors, warnings, and messages
BAR_RETRY        1                 # Number of times to retry failures
BAR_XPORT_COUNT  10                # Number of transport buffers per worker
BAR_XFER_BUFSIZE 8                 # Size of each transport buffer

LOG_BACKUP_MODE  MANUAL            # When to backup logical log files:
                                   # CONT - backup as soon as the file fills
                                   # MANUAL - user must perform log backups
                                   # NONE - don't backup logs and re-use them
                                   #        as soon as they fill.  Use of
                                   #        NONE is not recommended as data
                                   #        will be lost if the Dynamic Server
                                   #        must be restored
BAR_IDLE_TIMEOUT 5
BAR_BSALIB_PATH  /usr/lib/ibsad001.so  # full pathname of storage manager
                       # XBSA shared library
# Storage Manager instances
BAR_SM           1     # Storage manager ID
BAR_SM_NAME      A      # Storage manager name
BAR_WORKER_COSVR 1     # Storage manager is located on coserver 1
BAR_DBS_COSVR    1-5     # Route dbspaces from this coserver to this
                       # storage manager instance
BAR_LOG_COSVR    1-5     # Route logical log files from this coserver to this
                       # storage manager instance
BAR_WORKER_MAX   1     # Number of backup/restore objects this storage
                       # manager can support concurrently
END
```

Appendix

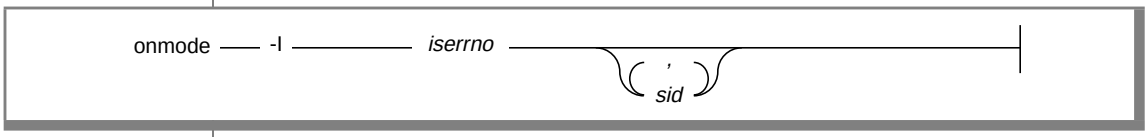
B

Trapping Errors

Occasionally, a series of events causes the database server to return unexpected error codes. If you do not have the appropriate diagnostic tools in place when these events occur, it might be difficult for you to determine the cause of these errors.

To help you collect additional diagnostics, you can use **onmode -I** to instruct Dynamic Server to perform the same diagnostics collection procedures described in "Collecting Diagnostic Information" on page 27-11. You can use **onmode -I** whenever you encounter an error number by supplying the *iserrno* and an optional session ID. The **-I** option is just one of many **onmode** options. For more information about **onmode**, see "onmode: Mode and Shared-Memory Changes" on page 35-28.

Syntax



Element	Purpose	Key Considerations
-I *iserrno*	Error number of the error for which you want to collect diagnostic information.	None.
sid	Session ID of the session for which you want to collect diagnostic information.	None.

Whenever the database server sets the *iserrno* to this value, the corresponding diagnostics events will occur producing an **af**.* file that you can fax or email to Informix Technical Support. b

Index

A

Adding listen threads 9-33
Adding virtual processors 9-17
ADM (Administration virtual processor) 9-17
Administrative tasks
  configuration tasks 2-7
  consistency checking 27-4
  controlling location of storage 13-16
  cron jobs 3-34
  initial tasks 2-4
  list of tasks 3-31
  planning 3-3
  routine tasks 2-5
  situations to avoid 28-3
  startup and shutdown scripts 3-31
ADT (audit) virtual processor 9-36
ADTMODE parameter
  mentioned 9-36
AFF_NPROCS parameter
  description of 33-9
  purpose of 9-22
AFF_SPROC parameter
  description of 3-24, 33-10
  processor affinity 9-23
  purpose of 9-22
af.xxx file A-4
Aggregate, parallel processing of 9-10
AIO virtual processors
  how many 9-28
  NUMAIOVPS parameter 9-27
  when used 9-27
ALARMPROGRAM parameter 29-8
Allocating disk space
  extent 13-14
  for mirrored data 24-5
  initial configuration 3-28
  procedure 14-4
ALTER TABLE statement
  mentioned 4-51
Alternate dbservername 33-17
ANSI compliance
  level Intro-16
ANSI-compliant transaction logging. *See* Logging.
Application
  client. *See* Client application.
Applications, types of 15-12
Archive
  preparing environment in multiple-residency setting 6-8
  reserved page information 38-14
  using ontape 35-95
  with multiple residency 6-8
Archive configuration file A-9
Archiving
  strategy 3-3
ARC_KEYPAD environment variable A-9
ASF 4-6
ASF. *See* Connectivity.
Assertion failure
  and data corruption 27-9
  description of 2-6, 27-3
  determining cause of 27-11
  DUMPCNT parameter 3-27, 33-32

DUMPCORE parameter 33-33
DUMPSHMEM parameter 3-27, 33-35
during consistency checking 27-7
during processing of user request 27-8
form in message log 27-7
Assertion failure file A-4
af.xxx 27-7
gcore A-5
gcore.xxx 27-7
list of 27-7
shmem.xxx 27-7, A-10
Asterisk, as wildcard in hostname field 4-36
Asynchronous I/O 9-26
and NUMAIOVPS parameter 33-61
write requests for mirrored data 23-9
Attaching to shared memory
additional segments 11-15
client to communications portion 11-11
database server utilities 11-12
description of 11-11
virtual processors 11-12
virtual processors and key value 11-13
Audit mode 9-36
Audit records
and sysaudit table 34-10
Auditing 1-12, 34-9, 34-10
Authentication
default policy 4-14
description of 4-14
Automatic recovery, by two-phase commit protocol 30-10
Availability
and critical media 13-16
as goal in efficient disk layout 13-33
improving with fragmentation 15-5
sample disk layout 13-40

B

Backup
active users, and 1-9
and fragmentation 15-5
dbspace backup, mentioned 1-9
displaying contents of 35-23
freeing a log file 19-14
mentioned 13-32
of TEXT and BYTE data 13-21
planning for 3-4
reducing size of 13-21
strategy 3-3
transaction records 1-9
with multiple residency 6-8
Bad-sector mapping, absence of 1-12
Before-image
contents 22-6
described 20-3
flushing of 11-47
in physical log buffer 11-47
journal. *See* Physical log.
role in buffer modification 11-46
Big buffers
and regular buffers 11-31
description of 11-31
Big-remainder page 38-41
Binding CPU virtual processors
and AFF_NPROCS parameter 33-9
benefit of 9-12
parameters 9-23
Bit-map page
2-bit values describing fullness 38-26
4-bit values describing fullness 38-27
component of a tblspace 13-27
component of an extent 38-29
component of the tblspace tblspace 38-22, 38-23
description of 38-25
location in extent 38-25
of a blobspace 38-64
role of 4-bit page 38-26
structure of 38-25
types of entries 38-25
Blobpage
average fullness statistics 35-13
blobpage size 38-59
blobpage size and storage efficiency 14-21
components of page header 38-66
description of 13-11
freeing deleted pages 18-22
fullness terminology explained 14-23
interpreting average fullness 14-23
oncheck -pB display explained 14-22
relationship to chunk 13-12
size, storage efficiency of 38-59
sizing for performance 14-21
sizing recommendations 13-12
specifying size of 38-59
storage statistics 14-21
structure and storage 38-59, 38-65
structure in dbspace 38-62
writes bypass shared memory 11-63
Blobspace
activating 18-21
adding a chunk 14-14, 14-18
bit-map page 38-64
blobpage structure 38-65
blobpage time stamps 11-61
buffers 11-64
creating using ON-Monitor 14-17
creating using onspaces 14-17
description of 13-21
dropping using ON-Monitor 14-20
dropping, initial tasks 14-19
free-map page
description of 38-64
location in blobspace 38-60
role in blobpage logging 18-23, 38-64
role in tracking blobpages 11-65
tracked by bit-map 38-64
in physical and logical log 13-21
logical-log administration tasks for 18-21
monitoring for fullness 13-21
multiple residency 6-8

overhead pages 18-22
page types 38-64
purpose of 13-21
restriction concerning dropping 35-52
storage efficiency 14-21
storage statistics 14-21
structure 38-59
structure of blobspace mirror chunk 38-17
TEXT and BYTE data storage 38-61
writing data to 11-63
writing TEXT and BYTE data 11-65
Blobspace blobpage
and physical logging 20-5
Block device 13-6
Boot file. *See* Startup script.
B-tree
cleaner list 38-57
indexing 38-48
structure 38-48
Buffer
access-level flag bits 35-71
big buffers 11-31
blobpage buffer 11-64
concurrent access 11-43
current lock-access level for 11-27
data replication 11-24, 25-10
dirty 11-46
exclusive mode 11-36
flushing 11-46
how a thread accesses a buffer page 11-43
how a user thread acquires 11-38
how thread releases after modification 11-46
least-recently used 11-38
lock types 11-36
lock-access level of 11-44
logical-log buffer 11-22, 11-46
maximum number 11-20
minimum requirement 11-20
monitoring statistics and use of 29-20
most-recently used 11-38
not dirty 11-46
page-type codes 35-70
physical-log buffer 11-24, 11-46
reading from disk 11-44
regular buffer 11-19
releasing if no thread waiting 11-45
releasing if thread waiting 11-45
releasing when modified 11-45
releasing when not modified 11-45
share lock 11-36
status 11-27
synchronizing flushing 11-46
threads waiting for 11-27
what occurs when modified 11-45
write types during flushing 11-51
Buffer flushing
description of 11-46
how synchronized 11-50
Buffer pool
bypassed by blobspace data 11-63
contents of 11-19
description of 11-19
flushing 11-47
LRU queues management 11-38
monitoring activity 29-23
read-ahead 11-42
synchronizing buffer flushing 11-50
Buffer table
contents of 11-27
description of 11-27
LRU queues 11-37
Buffered logging, when flushed 16-9
Buffered transaction logging. *See* Logging.
BUFFERS parameter
description of 11-20, 33-12
Buffer-size option, in options field 4-44
BUFFSIZE. *See* Page size.
buildsmi script
buildsmi.xxx file, description A-4
error log message 36-16
failure of A-4
when initializing database server 34-4
BYTE data type
Committed Read isolation 11-62
requires 4-bit bit map 38-26, 38-29
Byte lock 11-29

C

Cache. *See* Shared-memory buffer pool.
Cascading deletes 16-5
Cataloger, ON-Archive, mentioned 6-8
Cautions 28-3
Central registry, sqlhosts 4-29
Checkpoint
and chunk writes 11-53
and flushing of regular buffers 11-47
and logical-log file 11-57
and physical-log buffer 11-57, 20-11
description of 11-57
events that initiate 11-57
force with onmode -c 35-32
forcing, to free logical-log file 12-16
mentioned 8-9
monitoring activity 29-16
reserved page information 38-9
role in data replication 25-14
role in fast recovery 11-59, 22-5, 22-6
step in shared-memory initialization 8-9
updating the reserved pages 38-7
Chunk
activity during mirror recovery 23-8
adding to a blobspace 14-18
adding to a dbspace 14-14
adding to mirrored dbspace 24-9
adding using ON-Monitor 14-15
adding using onspaces 14-15
allocating initial 14-8
and associated partitions 13-34
changing mirror chunk status 24-10
checking for overlap 35-12

chunk status as PD 27-10
creating a link to the pathname 14-9, 24-5
defining multiple in a partition 14-5
description of 13-5
dropping from a blobspace or sbspace 14-19
dropping from a dbspace 14-18
exceeding size limits with LVM 13-46
free-list page 38-15, 38-16, 38-18
general disk layout guidelines 13-34
initial chunk of dbspace 38-4
initial mirror offset 33-53
I/O errors during processing 27-10
limits on size and number 13-5
maximum size of 13-5
mirror chunk reserved page information 38-13
monitoring 29-53, 34-11
multiple residency 6-8
name, when allocated as raw device 13-7
pathname stored 38-11
purpose of 13-5
recovering a down chunk 24-10
relation to extent 13-14
relinking after disk failure 24-12
reserved page information 38-11
structure
additional dbspace chunk 38-16
initial dbspace chunk 38-15
mirror chunk 38-17
using a symbolic link for the pathname 33-76
Chunk free list
how checked by oncheck 35-11
Chunk table
and mirroring 11-28
contents of 11-28
description of 11-28
Chunk write
checkpoints 11-53
monitoring 29-24
CKPTINTVL parameter
description of 33-13
initiating checkpoints 11-57
Classes of virtual processor 9-7
CLEANERS parameter
description of 33-14
purpose of 11-29
Client
and USEOSTIME configuration parameter 33-91
client id 35-77
connecting to primary server 25-5
connecting to secondary server 25-5
description of 1-4
designing for redirection 25-23
enabling communication 4-17
finding and connecting to a host 4-27
improving performance and availability 25-4
killing a client session 35-33
loopback connection 4-12
printing client sessions 35-76
reacting to failure 25-17
redirecting and connectivity in data replication 25-22
redirecting in data replication 25-18
remote 4-22
results of connection 33-62
security requirements 4-22
specifying a dbservername 4-51
specifying dbservername 33-18
Client application
redirecting in data replication 25-22
testing 5-4
wildcard addressing 4-38
Client-server protocol 4-9
Client/server architecture
description of 1-4
Client/server configuration
example
7.2 client with 5.x server 4-63
7.2 relay module 4-60
local loopback 4-55
multiple connection types 4-57
multiple database servers 4-59
multiple residency 4-59
network connection 4-56
shared memory 4-54
using IPX/SPX 4-57
listen and poll threads 9-30
local loopback 4-12
remote host 4-8
shared memory 4-10
Cold restore, number of recovery threads 33-63
Command-line conventions
elements of Intro-11
example diagram Intro-12
how to read Intro-12
Comment icons Intro-9
COMMIT statement, unbuffered logging 16-9
Committed Read isolation level
data-consistency checks 11-62
role of blob time stamps 11-61
Commit, heterogeneous 30-38
Communication configuration file. *See* ONCONFIG configuration file.
Communication support services
description of 4-13
Communications portion of shared memory
how client attaches 11-11
Communications portion (shared memory)
contents of 11-34
description of 11-34
size of 11-34
Communications support module
configuring 4-15
DCE-GSS module 4-14
description of 4-14
sqlhosts file 4-15
sqlhosts option field 4-45
Communications support services
message integrity 4-14
message privacy 4-14
Communication, client to database server. *See* Connectivity.
Compactness, of index page 33-36
Compliance, with industry standards Intro-16

concsm.cfg file
configuring the communications support module 4-15
format of entries 4-25
location of 4-16
options field for DCE-GSS 4-25
specifying the communications support module 4-16
Concurrency
improving with fragmentation 15-5
Concurrency control 11-35
Configuration
estimating required disk space 13-33
monitoring 29-14
overview 3-7
parameter overview 3-13
planning for database server 3-3
using 7.2 relay module 4-60
Configuration file
and multiple residency 5-3, 6-5
connectivity 4-17
reserved page information 38-8
warning about multiple residency 6-5
Configuration parameter
See also Configuration parameter use.
See also each parameter listed under its own name.
AFF_NPROCS 33-9
AFF_SPROC 3-24, 33-10
ALARMPROGRAM 29-8
BUFFERS 33-12
CKPTINTVL 33-13
CLEANERS 33-14
CONSOLE 3-12, 33-15
DBSERVERALIASES 4-51, 33-17
DBSERVERNAME 3-12, 4-31, 4-51, 33-18
DBSPACETEMP 33-19
DEADLOCK_TIMEOUT 33-21
DRAUTO 3-26, 33-22
DRINTERVAL 3-26, 33-23
DRLOSTFOUND 3-26, 33-24
DRTIMEOUT 3-26, 33-25
DS_MAX_QUERIES 33-26
DS_MAX_SCANS 33-27
DS_TOTAL_MEMORY 33-28
DUMPCNT 3-27, 33-32
DUMPCORE 3-27, 33-33
DUMPDIR 3-27, 33-34
DUMPGCORE 3-27, 33-34
DUMPSHMEM 3-27, 33-35
FILLFACTOR 33-36
HETERO_COMMIT 33-37
LOCKS 33-38
LOGBUFF 33-40
LOGFILES 33-41
LOGSIZE 33-42
LOGSMAX 33-43
LRUS 11-39, 33-44
LRU_MAX_DIRTY 3-21, 33-45
LRU_MIN_DIRTY 3-21, 33-45
LTAPEDEV 3-12
LTXEHWM 33-49
LTXHWM 33-50
MAX_PDQPRIORITY 33-51
MIRROR 33-52
MIRROROFFSET 33-53
MIRRORPATH 33-53
MSGPATH 3-12, 33-54
MULTIPROCESSOR 33-55
NETTYPE 4-50, 33-57
NOAGE 33-60
NUMAIOVPS 33-61
NUMCPUVPS 33-62
OFF_RECVRY_THREADS 3-25, 33-63
ON_RECVRY_THREADS 3-25, 33-64
OPTCOMPIND 33-67
PHYSBUFF 33-68
PHYSDBS 33-69
PHYSFILE 33-70
RA_PAGES 3-21, 33-71
RA_THRESHOLD 3-21, 33-72
RESIDENT 33-73
RESTARTABLE_RESTORE 33-74
ROOTOFFSET 33-75
ROOTPATH 3-12, 33-75, 33-76
ROOTSIZE 33-77
SERVERNUM 3-12, 11-12, 33-78
SHMBASE 11-12, 33-80
SHMVIRTSIZE 33-82
SINGLE_CPU_VP 33-83
STACKSIZE 33-84
STAGEBLOB 33-85
TAPEDEV 3-12
TXTIMEOUT 33-90
USEOSTIME 33-91
Configuration parameter use
and initial chunk of root dbspace 13-19
displayed
in data-replication screen 32-15
in diagnostics screen 32-16
in initialization screen 32-12
in PDQ screen 32-17
in performance screen 32-14
in shared-memory screen 32-13
enabling Logs Full HWM 18-9
for diagnostic information 27-11
for multiple residency 5-5
shared memory 12-3
Configuring
network card 4-19
TCP/IP network software 4-19
Configuring multiple ports 4-21
config.arc file
with multiple residency 6-8
CONNECT statement
example 4-52
mentioned 4-51
Connecting
and multiple residency 6-7
description of 9-30
methods 9-28
to multiple database servers 4-59
to multiple servers 5-3
to non-Informix databases 1-11
Connecting to the Windows NT Network application 4-19
server to server 4-9
Connection
7.2 client with 5.x server, example 4-63
database vs. network 4-7
IPX/SPX 4-57
local loopback, definition of 4-12
local loopback, example 4-55
multiple connection types example 4-57
multiple residency, example 4-59
multiplexed 4-7
network, description of 4-8

network, example 4-56
network, when to use 4-8
security restrictions 4-22
shared memory, description of 4-10
TCP/IP 4-22
Connectivity
configuration file 4-17
configuration parameters 4-50
file, sqlhosts 3-28
hosts file 4-19
services file 4-19
Consistency checking
corruption of data 27-9
data and overhead 27-4
index corruption 27-9
monitoring for data inconsistency 27-7
overview 27-3
periodic tasks 27-4
validating extents 27-6
validating indexes 27-6
validating reserved pages 27-5
when to schedule 27-4
Console messages 29-12
CONSOLE parameter
changing 29-12
description of 33-15
setting 3-12
Constraint, deferred checking 16-5
Contention. *See* Disk contention.
Context switching
description of 9-14
how functions when OS controls 9-10
OS versus multithreaded 9-10
Contiguous space for physical log 21-4
Control structures
description of 9-13
queues 9-16
session control block 9-13
stacks 9-15
thread control block 9-13
Conversion
during initialization 8-9
Cooked file space
and buffering 13-8
compared with raw space 13-7
contiguity of space 13-9
database server management of 13-7
description of 13-7
for static data 13-9
how to allocate 14-7
rationale for using 13-8
reliability 13-9
warning 13-7
Coordinating database server
and automatic recovery 30-11
description of 30-5
Coordinator recovery mechanism 30-11
Core dump
and DUMPCORE configuration parameter 33-33
contained in core file A-5
contents of gcore.xxx 27-7
when useful 27-11
See also DUMPCNT, DUMPDIR, DUMPGCORE, DUMPSHMEM.
core.pid.cnt file 33-35
Correlated subquery, effect of PDQ 15-22
Corruption
corrective actions 27-9
determining if exists 27-9
I/O errors from a chunk 27-10
symptoms of 27-9
CPU
binding to virtual processor 9-12
relationship to virtual processor 9-5
time tabulated 35-85
CPU virtual processor
adding and dropping in on-line mode 9-22
AFF_NPROCS parameter 9-22
AFF_SPROC parameter 9-22
and poll threads 9-29
and SINGLE_CPU_VP parameter 33-83
binding 33-9
description of 9-20
how many 9-20
on a multiprocessor computer 9-21
on a single-processor computer 9-21, 9-22
preventing priority aging 9-22
the NUMCPUVPS parameter 9-20
types of threads run by 9-20
CREATE INDEX statement
effect of PDQ 15-20
using FILLFACTOR 33-36
CREATE TABLE
TEMP 15-8
CREATE TABLE statement
mentioned 4-51
Critical dbspaces
mirroring 13-34
storage of 13-16
Critical media
mirroring 13-44
Critical section of code
and checkpoints 11-58
and filling of physical-log buffer 11-49
description of 11-56
related to size of physical log 20-5
cron jobs, warning about 3-34

D

Data block. *See* Page.
Data consistency
fast recovery 22-3
how achieved 11-56
monitoring for 27-7
symptoms of corruption 27-9
time stamps 11-60
verifying consistency 27-4
Data definition statements, when logged 16-6
Data files. *See* Logging.
Data management 13-7
Data manipulation statements, when logged 16-7

Data pages
and oncheck 35-10
Data replication
actions to take if primary fails 25-18
actions to take if secondary fails 25-18
administration of 26-15
advantages of 25-4
and the PAGE_ARCH reserved page 38-14
automatic switchover 25-19
changing database server mode 26-20
changing database server type 26-21
client redirection
comparison of different methods 25-33
handling within an application 25-30
using DBPATH 25-23
using INFORMIXSERVER 25-29
using sqlhosts file 25-25
configuring connectivity for 26-9
database and data requirements for 26-6
database server configuration requirements for 26-6
description of 1-10, 25-4
designing clients to use secondary database server 25-35
designing clients to use the primary server 25-34
detecting failures of 25-16
DRAUTO parameter 25-19
DRINTERVAL parameter 25-11
DRLOSTFOUND parameter 25-12
DRTIMEOUT parameter 25-16
flush interval 3-26, 33-23
hardware and operating system requirements for 26-5
how it works 25-8
how updates are replicated 25-9
importance of reliable network 25-21
information in sysdri table 34-17
initial replication 25-8
lost-and-found file 33-24
lost-and-found transactions 25-11
manual switchover 25-21
mentioned 16-5
monitoring status 29-71
planning for 26-4
possible failures 25-15
read-only mode 7-4
restarting after failure 25-20, 25-22, 26-26
restarting after network failure 26-29
restoring system after media failure 26-23
role of checkpoint 25-14
role of log records 25-9
role of primary database server 25-5
role of secondary database server 25-5
role of temporary dbspaces 25-38
setting up 26-5
specialized threads 25-13
starting 26-11
synchronization 25-14
wait time for response 3-26, 33-25
with asynchronous updating 25-11
with synchronous updating 25-11
Data row
and rowid 38-39
big-remainder page 38-41
forward pointer 38-39, 38-42
home page 38-37, 38-41
how database server locates 38-39
storage strategies 38-37
storing data on a page 38-40
TEXT and BYTE data descriptor 38-61
Data storage
control of 13-17
limits on due to maximum chunk size 13-5
overview 13-3
types of 13-3
See also Disk space.
Data Type segment
See also Disk space.
Database
controlling storage location 13-23
creating, what happens on disk 38-68
demonstration Intro-5
description of 13-22
effects of creation 38-68
estimating disk space for 13-33
estimating size of 13-33
information in sysdatabases table 34-14
locale, in sysdbslocale table 34-15
location of 13-22
migration. *See* Migration.
monitoring 29-43
owner, in sysmaster database 34-14
purpose of 13-22
recovery. *See* Recovery.
size limits 13-23
tuning. *See* Performance tuning.
Database administrator
See also Administrative tasks.
Database I/O 9-26
Database logging status
ANSI-compliant, description 16-10
buffered, description 16-9
changes permitted 17-3
changes, general info 17-4
changing buffering status
using ON-Archive 17-7
using ontape 17-8
description of 16-8
ending logging
using ON-Archive 17-6
using ontape 17-8
in a distributed environment 16-10
making ANSI-compliant
using ON-Archive 17-7
using ontape 17-9
modifying
using ON-Archive 17-5
using ON-Monitor 17-9
using ontape 17-7
setting 16-8

turning on logging
using ON-Archive 17-5
using ontape 17-7
unbuffered, description 16-9
who can change 16-11
Database schema. *See* Informix Migration Guide.
Database server
connecting to multiple 4-59, 5-3
multiple instances 5-3
remote 4-23, 34-27
security 1-12
shut down with UNIX script 3-32
Database server name. *See* dbservername.
DATABASE statement, mentioned 4-51
Database tblspace
entries 38-24
location in root dbspace 38-5, 38-24
relation to systable 38-69
structure and function 38-24
tblspace number 38-24
Data-recovery mechanisms
fast recovery 22-3
mirroring 23-4
Data-replication buffer 11-24, 25-9
DATASKIP parameter
specifying with onspaces utility 35-61
Data, estimating disk space for 13-31
DBNETTYPE environment variable 4-61
DBPATH environment variable
use in automatic redirection 25-23
DBPATH, and dbserver group 4-48
Dbserver group
purpose of 4-46
DBSERVERALIASES parameter
and multiple residency 5-5
description of 4-51, 33-17
example 4-51
in sqlhosts file 4-31
multiple connection types example 4-57
dbservername
description of 33-18
field in sqlhosts file 4-31
syntax rules 4-31
DBSERVERNAME parameter
and multiple residency 5-5
associated protocol 9-29
description of 4-51, 33-18
in sqlhosts file 4-31
multiple residency 6-6
setting 3-12
virtual processor for poll thread 9-29
Dbspace
adding a chunk to 14-14
adding a mirrored chunk 24-9
as link between logical and physical units of storage 13-16
bit-map page 38-25
blobpage structure 38-62
blobpage time stamps 11-61
creating a temporary 14-12
creating during initial configuration 3-30
creating with ON-Monitor 14-13
creating with onspaces 14-13
creating, overview 14-11
description of 13-16
dropping a chunk from 14-18
dropping using ON-Monitor 14-20
dropping, overview 14-19
identifying the dbspace for a table 38-22
initial dbspace 13-19
list of structures contained in 38-15
mirror chunk information 38-7
mirroring if logical log files included 23-7
modifying with onspaces utility 35-61
monitoring TEXT and BYTE data 29-69
monitoring with SMI 34-15
multiple residency 6-8
page header 38-34
primary chunk information 38-7
purpose of 13-16
reserved page information 38-11
role in fragmentation 15-3
root dbspace defined 13-19
root name 33-75
shared-memory table 11-28
starting to mirror 24-8
storage 38-4
structure 38-4, 38-15, 38-16
structure of additional dbspace chunk 38-16
structure of chunk free-list page 38-18
structure of mirror chunk 38-17
structure of nonroot dbspace 38-15
structure of tblspace tblspace 38-19
temporary 13-20
TEXT and BYTE data storage 38-61
usage report 38-5
Dbspace table
contents of 11-28
description of 11-28
Dbspaces
creating 14-8
DBSPACETEMP environment variable 13-27
DBSPACETEMP parameter
and load balancing 13-35
description of 33-19
if not set 13-27
relationship to DBSPACETEMP environment variable 13-26
DCE components
installing and configuring 4-15
DCE-GSS communications support module
configuring 4-15
description of 4-14
shared library 4-14
shared library location 4-14
sqlhosts 4-15
steps to use 4-15
Deadlock prevention 18-8
Deadlock. *See* DEADLOCK_TIMEOUT parameter.

DEADLOCK_TIMEOUT parameter
description of 33-21
in two-phase commit 30-37
Decision-support application
characteristics 15-15
definition of 15-14
uses of 15-14
Decision-support query
DS_MAX_QUERIES parameter 3-23, 33-26
DS_TOTAL_MEMORY parameter 33-28
MAX_PDQPRIORITY parameter 33-51
monitoring resources allocated for 29-38
monitoring threads for 29-37
parameters, setting with onmode 35-39
See also PDQ.
Default configuration file 8-7
Default locale Intro-4
Deferred checking of constraints 16-5
Delete flag
described 38-49
Demonstration database Intro-5
Descriptor
TEXT and BYTE data 38-61
Detached index 15-8
Device 13-6, 13-7
Devices
when are offsets needed 14-5
Diagnostic information
and disk space restraints 27-11
collecting 27-11
parameters to set 27-11
Diagnostic messages. *See* Message log.
Diagnostics
using onmode B-1
Dictionary cache 11-33
Dirty buffer, description of 11-46
Dirty Read isolation level
data-consistency checks 11-62
role of blob time stamps 11-61
Disabling I/O error
circumstances under which they occur 13-15
defined 13-15
destructive versus nondestructive 13-15
monitoring using event alarms 29-53
monitoring using message log 29-52
Disk contention
and high-use tables 13-37
of critical media 13-35
reducing 13-33
Disk failure 1-9
Disk I/O
kernel asynchronous I/O 9-24
logical log 9-24
operating system I/O 13-7
physical log 9-24
priorities 9-25
raw I/O 13-6
reads from mirrored chunks 23-9
role of shared memory in reducing 11-5
virtual processor classes 9-24
writes to mirrored chunks 23-9
Disk I/O queues 9-28
Disk layout
and archiving 13-36, 13-38
and logical volume managers 13-46
and mirroring 13-38
and table isolation 13-37
for optimum performance 13-34
sample disk layouts 13-40
trade-offs 13-40
Disk page
before-images in physical log 11-47
function of time stamp pairs 11-60
logical page number 38-38
number to read ahead 3-21, 33-71
page compression 38-36, 38-46
physical page number 38-38
read ahead 11-42
storing data on a page 38-40
structure
blobspace blobpage 38-29
dbspace page 38-34
types of pages in an extent 38-29
Disk space
allocating
raw disk space 13-8
when a database is created 38-68
when a table is created 38-70
allocating cooked file space 14-7
allocating raw disk, Windows NT 14-9
allocation for system catalogs 38-68
caution with multiple residency 6-8
chunk free-list page 38-18
configuring for multiple residency 5-6
creating a link to chunk pathname 14-9
described 14-4
estimating size of 13-30
initialization 8-8
definition of 8-3, 14-10
with new database server 3-30
with oninit 14-11, 35-20
with ON-Monitor 14-11
layout guidelines 13-33
limits on due to maximum chunk size 13-5
list of structures 38-3
locating free space 38-18
multiple residency 6-7
offsets for chunk pathnames 14-5
optimal storage of tables 13-39
optimizing temporary space layout 13-35
page compression 38-36, 38-46
raw devices versus cooked files 13-6
requirements 13-33
for root dbspace 13-30
storage of TEXT and BYTE data 14-22

strategies for improving performance 13-33
temporary. *See* Temporary disk space.
tracking
available space in a blobspace 38-64
available space in a chunk 38-18
free pages with bit-map page 38-25
usage by tblspace 13-27
Distributed databases 1-11
Distributed environment, logging status 16-10
Distributed queries, description of 1-11
Distributed transaction
and two-phase commit protocol 30-3
determining if inconsistently implemented 31-6
mentioned 16-5
Distribution scheme 15-6
Documentation
on-line notes Intro-15
Documentation conventions
command-line Intro-10
icon Intro-8
typographical Intro-8
Documentation notes Intro-15
Documentation notes, program item Intro-15
Documentation, types of
documentation notes Intro-15
error message files Intro-14
machine notes Intro-15
on-line manuals Intro-13
printed manuals Intro-14
release notes Intro-15
Domain, Windows NT
and database server 4-5
and user accounts 4-5
controller 4-5
description of 4-5
trusted 4-6
user account in 4-6
DRAUTO parameter
description of 3-26, 33-22
role in recovering from data-replication failure 25-19
set to RETAIN_TYPE 25-20
set to REVERSE_TYPE 25-20
DRINTERVAL parameter
description of 3-26, 33-23
setting for asynchronous updating 25-11
setting for synchronous updating 25-11
DRLOSTFOUND parameter
description of 3-26, 33-24
use with data replication 25-12
DRTIMEOUT parameter
description of 3-26, 33-25
role in detecting data replication failures 25-16
dr.lostfound file 33-24
DSA. *See* Dynamic scalable architecture.
DSS applications. *See* Decision-support applications.
DS_MAX_QUERIES parameter
description of 3-23, 33-26
DS_MAX_SCANS parameter
description of 3-23, 33-27
DS_TOTAL_MEMORY parameter
deriving the value 33-30
description of 3-23, 33-29
recalculated by server 33-29
DUMPCNT parameter 3-27, 33-32
DUMPCORE parameter 3-27, 33-33
DUMPDIR parameter
af.xxx assertion failure file A-4
and consistency checking 27-7
and shmem file A-10
description of 3-27, 33-34
gcore file A-5
DUMPGCORE parameter 3-27, 27-7, 33-34
DUMPSHMEM parameter 3-27, 27-7, 33-35
Dynamic Host Configuration Product 4-20
Dynamic scalable architecture
advantages 9-3
description of 9-3
Dynamic Server
bad-sector mapping, absence of 1-12
client/server architecture 1-4
distributed queries 1-11
fault-tolerant features 1-8
features beyond the scope of 1-12
high performance of 1-6
scalability of 1-5
TEXT and BYTE data compression, absence of 1-13

E

Enterprise Replication 4-46
Environment configuration file A-6
Environment variable
ARC_KEYPAD A-9
DBNETTYPE 4-61
for users of client applications 3-33
INFORMIXCONCSMCFG 4-16
INFORMIXDIR 3-8, 3-32, 4-61
INFORMIXSERVER 3-9
INFORMIXSHMBASE 11-11
INFORMIXSQLHOSTS 4-27
INFORMIXTERM 3-10
ONCONFIG 3-9
PATH 3-8, 3-32
PDQPRIORITY 15-17
setting 3-8
SQLEXEC 4-61
SQLRM 4-61
SQLRMDIR 4-61
TBCONFIG A-7
TERMINFO 3-10
en_us.8859-1 locale Intro-4
Error messages
files Intro-14
for two-phase commit protocol 30-29
I/O errors on a chunk 27-10
/etc/hosts file 4-17
and client redirection 25-27
/etc/services file 4-17
and client redirection 25-27
multiple residency 6-7

Event alarm
class ID parameter 29-10
class message parameter 29-10
description 29-8
event severity codes 29-9
Event alarms
mentioned 34-7
Example
7.2 relay module 4-60
connecting 7.2 client to 5.0 server 4-64
DBSERVERALIASES and sqlhosts file 4-52
DBSERVERALIASES parameter 4-51
/etc/services file entry 4-18
how page cleaning begins 11-41
IPX/SPX connection 4-57
local loopback connection 4-55
multiple connection types 4-57
NETTYPE parameters for tuning 33-58
relay module with three servers 4-62
shared-memory connection 4-54
SQLEXEC environment variable 4-61
TCP/IP connection 4-56
Exclusive lock (buffer), description of 11-36
exit codes, ontape utility 35-97
Explicit temporary table 13-26
Expression-based distribution scheme 15-7
Extent
automatic doubling of size 38-31
default size 38-28
description of 13-13
disk page types 38-29
how database server allocates 13-14
information in sysextents table 34-18
initial size 13-14, 38-28
key concepts concerning 13-14
merging 38-32
monitoring 29-60
next-extent allocation 38-31
next-extent allocation strategies 38-33
next-extent size 13-14, 38-31
procedure for allocating 38-31
purpose of 13-13
relationship to chunk 13-14
size limitations 38-28
structure 13-14, 38-27
tracking free pages using bit-map page 38-25
validating consistency 27-6
Extent size
index fragments 38-28
External spaces. *See* extspace.

F

Fast recovery
description of 1-9, 22-3
details of process 22-5
effects of buffered logging 22-4
how database server detects need for 22-4
mentioned 7-4, 8-9, 16-4
no logging 22-5
purpose of 22-4
role of checkpoint 11-59
role of PAGE_CHKT reserved page 22-6
when needed 22-4
when occurs 22-3
Fault tolerance
backups, and 1-9
data replication 1-10, 25-4, 25-6
fast recovery 1-9, 22-3
mirroring 1-10
Feature icons Intro-9
Features, product Intro-6
File
af.xxx A-4
buildsmi.xxx A-4
config.arc 6-8, 6-9
connectivity configuration 4-17
core A-5
core.pid.cnt 33-35
dr.lostfound 33-24
gcore 3-27, 33-34
gcore, xxx A-5
hosts.equiv 4-22
in INFORMIXTMP directory A-6
.informix A-6
informix.rc environment file A-6
.infos.dbservername A-7
.inf.servicename A-6
network security 4-22
oncatlgr.out.pidnum A-7
oncfg_servername.servernum 8-10, A-8
onconfig A-8
onconfig, during initialization 8-6
onconfig.std
description of A-8
during initialization 8-6
sample file A-10
oper_deflt.arc 6-8, 6-9, A-9
private environment file A-6
servicename.exp A-9
servicename.str A-9
shmem.pid.cnt 33-35
shmem.xxx A-10
sqlhosts 3-28
summary of database server files A-1
template for configuration file A-8
VP.servername.nnx A-10
File I/O. *See* Disk I/O.
Files
NTFS 13-6
FILLFACTOR parameter
control how indexes fill 38-58
description of 33-36
finderr script Intro-14
FLRU queues
and reading a page from disk 11-44
and releasing buffer 11-45
description of 11-37
See also LRU queues.
Flushing
buffers 11-46
data-replication buffer, maximum interval 3-26, 33-23
of before-images 11-47
Forced residency
initialization 8-10
start/end with onmode 35-32

Forcing a checkpoint. *See* Checkpoint.
Foreground write
  and before-image 11-47
  description of 11-52
  monitoring 11-52, 29-24
Forward pointer
  description of 38-39
  how it can become invalid 11-61
  role in a blobspace blobpage 38-65
  role in data storage 38-42
  role in dbspace blob storage 38-61
  unaffected by page compression 38-36
Fragment
  description of 15-3
  monitoring disk usage 29-60, 29-63
  monitoring I/O requests for 29-55
  turning DATASKIP ON or OFF for 35-61
  warning returned when skipped during query 33-16
  See also Fragmentation.
Fragmentation
  and memory-resident tables 11-21
  and mirroring 13-38
  and rowids 38-39
  description of 15-3
  distribution schemes for 15-6
  for improved concurrency 13-38
  goals of 15-5, 15-7
  of temporary tables 15-8
  strategy 15-6
  with PDQ 15-5
  See also Fragment.
Fragmented table, using primary keys 38-40
Fragments
  index 38-28
Free list. *See* Chunk free list.
Free-map page
  description of blobspace free-map page 38-64
  role in blobspace logging 18-23

G

Gateway transactions 33-37
Gateway, Informix, in heterogeneous commit 30-38
gcore
  file 27-7
  utility 3-27, 33-32, 33-34
gcore.xxx file A-5
Global Language Support (GLS) Intro-4
  mentioned 3-29
Global pool, description of 11-34
Global transaction identification number 31-7, 31-8
Group, parallel processing of 9-10

H

Hash table
  to buffer table 11-27
  to lock table 11-29
Hash tables 11-26
Heaps 11-33
Heterogeneous commit 30-38 to 30-46
Heterogeneous-commit transactions 33-37
HETERO_COMMIT configuration parameter 33-37
Heuristic decision
  result from independent action 30-20
  types of 30-21
Heuristic end-transaction
  conditions for 30-26
  description of 30-26
  determining effects on transaction 31-4
  illustration, including logical log records 30-35
  messages returned by 30-28
  results of 30-28
  when necessary 30-26
Heuristic rollback
  conditions resulting in 30-22
  description of 30-22
  illustration, including logical log records 30-33
  indications that rollback occurred 31-5
  results of 30-23
High-availability data replication. *See* Data replication.
HKEY_LOCAL_MACHINE registry 3-28
Home directory, Windows NT 4-23
Home page 38-37, 38-41
Horizontal index link
  described 38-55
  how implemented 38-56
hostname
  syntax rules 4-34, 4-40
hostname field
  multiple network interface cards 9-34
  using IP addresses 4-35
  wildcard addressing 4-36
  with IPX/SPX 4-40
  with shared memory 4-34
  with TCP/IP communication protocol 4-34
hosts file 4-19
  multiple residency 6-7
  on UNIX 4-18
  on Windows NT 4-20
hosts.equiv file 4-22
  sqlhosts security options 4-49
Hot site backup. *See* Data replication.

I

Icons
  comment Intro-9
  feature Intro-9
  platform Intro-9
  product Intro-9
illlsrra.xx file A-5
Impersonate, client 4-24
Inconsistency information
  how to detect 27-4

Index
branch node 38-48
btree cleaner list 38-57
controlling how filled 38-58
delete flag 38-49
duplicate key values 38-54
effects on structure of item insertion 38-56
horizontal node links 38-56
how created and filled 38-50
how deleted items removed 38-57
insertion of indexed data 38-56
item described 38-49
key value locking 38-57
leaf node 38-48
location of index page 38-55
parallel building of 9-10
physical format and storage 38-55
repairing structures with oncheck utility 35-5
reuse of freed pages 38-57
root node 38-48
structure of B+ tree 38-48
validating consistency 27-6
Index fragments
extent size 38-28
Index item
calculating the length of 38-58
defined 38-49
example of 38-50
how removed 38-57
merging 38-57
shuffling 38-58
when purged 38-49
Index page
compactness 33-36
contents 38-55
creation of first 38-51
effects of creation 38-51
effects of inserting item 38-56
page header 38-55
sibling pointers 38-55
size 38-55
structure 38-55
structure of 38-48
Industry standards, compliance with Intro-16
Infinity item
and creation of branch node 38-53
defined 38-52
.informix file
mentioned A-6
multiple residency 6-9
Informix Find Error utility Intro-14
Informix recommendations
on allocation of disk space 13-9
on consistency checking 27-4
on mirroring the physical log 20-8
INFORMIXCONCSMCFG environment variable 4-16
$INFORMIXDIR/etc/sqlhosts. *See* sqlhosts file.
INFORMIXDIR environment variable 3-8
and 5.x products 4-61
in shutdown script 3-32
in startup script 3-32
multiple residency startup script 6-9
INFORMIXDIR/bin directory Intro-5
INFORMIXKEYTAB environment variable
Environment variable INFORMIXKEYTAB 4-16
INFORMIXSERVER environment variable
and dbserver group 4-48
during initialization 3-9
multiple residency startup script 6-9
multiple versions of database server 3-32
relation to DBSERVERNAME 33-18
use in client redirection 25-29
values of 3-9
with multiple residency 6-9
INFORMIXSHMBASE environment variable 11-11
INFORMIXSQLHOSTS environment variable 4-27, 4-29
INFORMIXSTACKSIZE environment variable 11-32
INFORMIXTERM environment variable 3-10
INFORMIXTMP directory
description of A-6
servicename.exp file A-9
servicename.str file A-9
.inf.servicename file A-6
informix.rc environment file
description of A-6
multiple residency 6-9
use of 3-9
.infos.dbservername file
description of A-7
regenerate 35-38
.inf.servicename file A-6
Initial configuration
creating blobspaces and dbspaces 3-30
disk layout 13-33
guidelines for root dbspace 13-19
raw disk devices versus cooked files 13-6
Initialization
checkpoint 8-9
commands 14-10
configuration changes 8-9
configuration files 8-6
control returned to user 8-11
conversion of internal files 8-9
disk space 3-30, 8-3, 8-8
disk space for multiple residency 6-7
disk structures initialized 38-5
fast recovery 8-9
forced residency 8-10
message log 8-11
oncfg_servername.servernum file 8-10
onconfig file 8-6
onconfig.std file 8-6
reserved page information 38-8
shared memory 8-3
SMI tables 8-11
steps in 8-4
temporary tablespaces 8-10
upon completion 8-12
utilities for 8-4
virtual processors 8-8
INSERT cursor 15-6

Installation
  definition of 3-7
  when no other Informix products are present 3-7
Installing, TCP/IP network software 4-19
Integrity, data. *See* Consistency checking.
Interprocess communication
  in nettype field 4-32
  shared memory for 11-5
Interval, checkpoint 33-13
IP address
  how to find 4-35
  use in hostname field 4-35
ipcshm protocol and communications portion (shared memory) 11-34
IPC. *See* Interprocess communications.
IPX/SPX
  in hostname field 4-40, 4-57
  in servicename field 4-42
  multiple residency 6-7
  service, definition of 4-42
  sqlhosts entry 4-57
ISAM calls tabulated 35-85
ISO 8859-1 code set Intro-4
Isolation level
  Committed Read and blob time stamps 11-61
  Dirty Read and TEXT or BYTE data 11-61
I/O errors during processing 27-10
I/O. *See* Disk I/O.

J

Join, parallel processing of 9-10

K

KAIO thread 9-27
Keep-alive option, in options field 4-48
Kernel asynchronous I/O
  description of 9-27
  non-logging disk I/O 9-24
Key value
  and index items 38-49
  checking ordering with oncheck 35-11
  duplicates 38-54, 38-55
  for shared memory 11-13
  locking 38-57
keytab file
  purpose of 4-16

L

Latch
  and wait queue 9-18
  description of 11-35
  displaying information with onstat 35-67, 35-88
  identifying the resource controlled 35-88
  monitoring statistics and use 29-27
  mutex 9-19, 11-35
Level-0 dbspace
  use in consistency checking 27-8
Limits
  on chunk size 13-5
Linking, name of root dbspace 33-76
LIO virtual processors
  description of 9-25
  how many 9-25
Listen threads
  and multiple interface cards 9-34
  description of 9-30
Load balancing
  as performance goal 13-33
  done by virtual processors 9-9
  of critical media 13-35
  through use of DBSPACETEMP 13-35
Local loopback
  compared with shared-memory connection 4-12
  connection 4-12, 9-29
  example 4-55
  restriction 4-12
Locale Intro-4
Lock
  and oncheck 35-7
  and wait queue 9-18
  buffer-access-level flag bits 35-71
  description of 11-35
  information in syslocks table 34-18
  key value locking 38-57
  maximum time to acquire 33-21
  monitoring 29-29, 35-66, 35-79
  type codes 35-80
  types 11-36
Lock table
  contents of 11-29
  hash table 11-29
  maximum number of entries 11-29
Lock-access level, of a buffer 11-44
Locking
  for multiprocessor 33-55
  when occurs 11-44
  when released 11-45
LOCKS parameter
  description of 33-38
LOGBUFF parameter
  and logical log buffers 11-23
  description of 33-40
LOGFILES parameter
  changing 19-10
  description of 33-41
  use in logical-log size determination 18-7
Logging
  activity that is always logged 16-6
  database server processes requiring 16-4
  definition of transaction logging 16-8
  effect of buffering on logical log fill rate 18-19
  monitoring activity 29-44
  physical logging
    description of 20-3
    process of 20-9
    purpose of 20-4
    sizing guidelines for 20-5
    suppression in temporary dbspaces 13-21
  process for blobspace data 18-27

process for dbspace data 18-25
role in data replication 25-9
role of blobspace free-map page 18-28, 38-64
suppression for implicit tables 13-20
when to buffer transaction logging 16-11
when to use transaction logging 16-10
with TEXT and BYTE data 16-7
See also Database-logging status.
Logical consistency, description of 22-5
Logical log
administration tasks for blobspaces 18-21
backing up 18-8
checking consistency 35-12
configuration parameters 19-10
description of 11-22, 16-4, 18-3
determining disk space allocated 18-7
how to prevent logs full 18-8
in root dbspace 38-5
maximum number of files 33-43
monitoring for fullness using onstat 29-44
monitoring with SMI 34-19
optimal storage of 13-35, 13-36
preserving space in 18-8
purpose of 1-9
setting high-water marks 18-20
size, guidelines 18-6
size, performance considerations 18-5
types of records 11-22
See also Logical-log buffer, Logical-log file.
Logical page number 38-38
Logical recovery
number of threads 3-25, 33-64
role in data replication 25-9
Logical units of storage
description of 13-16
list of 13-4
Logical volume manager (LVM)
description of 13-46
mirroring alternative 23-6
Logical-log buffer
and data-replication buffer 11-24
and LOGBUFF parameter 33-40
and logical-log records 11-53
description 11-22
description of 11-22
flushing 11-53
flushing for non-logging databases 11-55
flushing when a checkpoint occurs 11-55
flushing when no before-image 11-56
flushing with unbuffered logging 11-55
monitoring 29-49
role in logging process 16-9
synchronizing flushing 11-46
when flushed to disk 11-53, 18-27
when it becomes full 11-54
Logical-log file
adding a log file
using ON-Monitor 19-5
using onparams 19-5
allocating disk space for 18-5
and reserved pages 38-9
backup
effect on performance 18-5
goals of 18-15
to free deleted blobpages 18-22
changing the size of 19-9
consequences of not freeing 18-16
created during initialization 33-41
description of 16-4, 18-4
displaying contents 35-23
dropping a log file
using ON-Monitor 19-7
using onparams 19-7
file status 18-13
how to free 19-13
how to switch 19-15
I/O to 9-24
location 18-12
logid number 18-12
mirroring a dbspace containing a file 23-7
moving to another dbspace 19-8
number of files 18-11
rate at which files fill 18-18
reading the log file 35-23
relationship between unique ID and logid 18-12
reserved page information 38-9
role in fast recovery 22-5, 22-7 to 22-8
size 18-11, 33-42
status flags 18-13
switch using onmode 35-33
switching to activate blobspace chunks 18-22
switching to activate blobspaces 18-21
unique ID number 18-12
when database server tries to free files 18-16
when freed for reuse 18-16
See also Logical log.
Logical-log I/O virtual processors 9-25
Logical-log record
additional columns of 37-7
database server processes requiring 16-4
displaying 35-23
flushing under two-phase commit protocol 30-32
for a checkpoint 37-5
for a drop table operation 37-4
generated by a rollback 37-4
header columns 37-6
involved in distributed transactions 37-5
involved in two-phase commit protocol 30-30, 37-5
role in fast recovery 22-7
role in two-phase commit protocol 30-6
SQL statements generating 16-7
types 37-7
when written to logical-log buffer 18-26
logid 18-12
Logs-Full High-Water Mark 18-8
LOGSIZE parameter
changing 19-10
description of 33-42

LOGSMAX parameter
changing 19-12
description of 33-43
mentioned 18-11
Long transaction
consequences of 18-17
description of 18-18
mentioned within two-phase commit discussion 30-19, 30-23, 30-26
preventing development of 18-18
starting percentage 33-49, 33-50
LRU queues
and buffer pool management 11-38
and buffer table 11-20
components 11-37
composition of 11-37
description of 11-37
displaying information with onstat 35-66, 35-87
FLRU queues 11-37
how database server selects 11-38
MLRU queues 11-37
modified pages, percentage of 3-21, 33-45
ordering of 11-38
rationale for ordering 11-38
reason for multiple 11-39
recommendation on configuring 11-39
LRU write
description of 11-52
monitoring 29-24
triggering of 11-52
who performs 11-52
LRUS configuration parameter
setting the value of 11-39
LRUS parameter
description of 11-37, 33-44
LRU_MAX_DIRTY parameter
and LRU_MIN_DIRTY parameter 11-37
description of 3-21, 33-45
example of use 11-41
how to calculate value 11-41
purpose of 11-40
role in buffer-pool management 11-40
LRU_MIN_DIRTY parameter
and LRU_MAX_DIRTY parameter 11-37
default value 11-41
description of 3-21, 33-45
example of use 11-41
how to calculate value 11-41
role in buffer pool management 11-41
when tested 11-51
LTAPEBLK parameter
description of 33-46
LTAPEDEV parameter
setting 3-12
LTXEHWM parameter
and physical log 20-8
changing 19-12
description of 33-49
role in heuristic rollback 30-22
role in preventing long transactions 18-20
LTXHWM parameter
changing 19-12
description of 33-50
flushing of logical-log buffer 11-54
role in heuristic rollback 30-22
role in preventing long transactions 18-20

M

Machine notes Intro-15
Main_loop() thread 11-58
role in checkpoint 11-59
role in two-phase commit recovery 30-12
Management of data 13-7
Manual recovery
deciding if action needed 31-9
determining if data inconsistent 31-6
example of 31-10
obtaining information from logical log files 31-7
procedure to determine if necessary 31-3
use of GTRID 31-7
Mapping, bad sector 1-12
MAX_PDQPRIORITY parameter
description of 33-51
Media failure
detecting 23-10
recovering from 23-5
restoring data 1-9
Memory
specifying size of 3-28, 33-66
Memory-resident tables
adding a fragment to 11-21
description of 11-21
Memory. *See* Shared memory.
Message log
alphabetical listing of messages 36-3
and data corruption 27-9
and event alarms 29-8
categories of messages 36-4
description of 29-7, 36-3
displaying with onstat utility 35-66, 35-82
during initialization 8-11
file pathname 33-54
location of 33-54
mentioned A-7
monitoring 29-7
Messages, for onspaces 36-31
MGM. *See* Memory Grant Manager.
Migration. *See Informix Migration Guide.*
Mirror chunk
adding 24-9
changing status of 24-10
creating 24-6
disk reads from 23-9
disk writes to 23-9
pathnames 38-13
recovering 23-10, 24-10
relinking after disk failure 24-12
structure 23-11, 38-17
Mirror dbspace, creating 14-8
MIRROR parameter
changing 24-4
description of 24-4, 33-52
initial configuration value 24-4

Mirroring
activity during processing 23-9
alternatives 23-6
and chunk table 11-28
and multiple residency 6-7
asynchronous write requests 23-9
benefits of 23-4
changing chunk status 24-10
costs of 23-5
creating mirror chunks 24-6
description of 1-10, 23-4
detecting media failures 23-10
during system initialization 24-7
enable flag 33-52
enabling 24-4
ending 24-12
if the dbspace holds logical-log files 23-7
initial chunk 33-53
network restriction 23-4
recommended disk layout 13-38
recovering a chunk 24-10
recovery activity 23-8
reserved page information 38-13
split reads 23-9
starting 24-3, 24-6
status flags 23-8
steps required 24-3
what happens during processing 23-9
when mirroring begins 23-7
when mirroring ends 23-11
MIRROROFFSET parameter 13-19
and multiple residency 5-5
description of 33-53
setting 24-7
when needed 14-6
MIRRORPATH parameter
and multiple residency 5-5
description of 33-53
mentioned 13-19
multiple residency 6-7
setting 24-7
specifying a link pathname 33-53
Miscellaneous (MSC) virtual processor 9-36
Mixed transaction result 30-25
MLRU queues
and flushing of regular buffers 11-47
and LRU_MIN_DIRTY parameter 11-41
description of 11-37
how buffer is placed 11-39
how to end page-cleaning 11-41
limiting number of pages 11-40
role in buffer modification 11-46
when cleaning ends 11-41
See also LRU queues.
Mode
description of 7-3
graceful shutdown 7-7
immediate shutdown 7-7
off-line from any mode 7-8
off-line to on-line 7-6
off-line to quiescent 7-5
on-line to quiescent, gracefully 7-7
on-line to quiescent, immediately 7-7
quiescent to on-line 7-6
reinitializing shared memory 7-5
taking off-line 7-8
users permitted to change 7-5
MODE ANSI keywords, and database logging status 16-10
Monitoring database server
active tblspaces 29-32
blobspace storage efficiency 14-21
buffer-pool activity 29-23
buffers 11-20, 29-20
checkpoints 29-16
chunks 29-53
configuration parameter values 29-14
databases 29-43
data-replication status 29-71
disk I/O queues 9-28
extents 29-60
fragment load 29-55
fragmentation disk use 29-60, 29-63
latches 29-18, 29-27
list of tools 3-5
locks 29-29
logging status 29-43
logical-log buffers 29-49
logical-log files 29-44
message log 29-7
PDQ resources 29-40
physical-log buffer 11-24, 29-49
physical-log file 29-47
profile of activity 29-19
sessions 29-35
shared memory 29-18
shared-memory segments 29-18
sources of information 29-6
stack size 29-38
TEXT and BYTE data in dbspaces 29-69
threads 29-35
transactions 29-41
user threads 11-31
using oncheck 29-12
using ON-Monitor 29-12
using onperf 29-13
using onstat 29-12
using SMI tables 29-12
virtual processors 29-33
Monitoring, length of disk I/O queues 9-28
MSGPATH parameter
and multiple residency 5-5
description of 33-54
multiple residency 6-6
setting 3-12
Multiple concurrent threads (MCT) 9-13
Multiple connection types
example 4-57
in sqlhosts file 4-52
See also Connection.
Multiple instances of database server on same computer 5-3
Multiple network interface cards 9-34
Multiple residency
and blobspace 6-8
and chunk assignment 6-8
and dbspaces 6-8
and multiple binaries, warning 6-4
archiving 6-8
backups 6-8

benefits of 5-3
configuration file 5-3
configuration parameters 5-5
DBSERVERNAME parameter 6-6
definition of 3-7, 5-3
editing the ONCONFIG file 6-5
example 4-59
hosts & services files 6-7
how it functions 5-4
INFORMIXSERVER environment variable 6-9
informix.rc & .informix files 6-9
initializing disk space 6-7
IPX/SPX 6-7
MIRRORPATH parameter 6-7
MSGPATH parameter 6-6
ONCONFIG environment variable 5-5
planning for 6-3
requirements 6-3
ROOTNAME parameter 6-6
ROOTPATH parameter 6-6
SERVERNUM parameter 6-5, 6-6
sqlhosts file 6-7
startup script 6-9
to isolate applications 5-4
use for testing 5-4
Multiplexed connection 4-7
description of 4-7
limitations 4-7
Multiprocessor computer
advantages on 9-3
AFF_SPROC parameter 33-10
MULTIPROCESSOR parameter 9-21
processor affinity 9-12
MULTIPROCESSOR parameter
description of 9-21, 33-55
for single-processor computer 9-21
Multi-threaded database server. *See* Dynamic scalable architecture.
Multithreaded processes, description of 9-5
Multithreading, use of OS resources 9-10
Mutex
description of 9-19, 11-35
on buffer table hash table 11-44
synchronization 9-20
when used 11-35

N

Named pipe connection
description of 4-11
mentioned 4-8
nettype field
description of 4-31
format of 4-31
summary of values 4-33
syntax of 4-31
use of interface type 4-56
NETTYPE parameter
and multiple network addresses 9-34
description of 33-57
for multiplexed connection 4-7
mentioned 4-50
ON-Monitor screen entries 10-6
purpose of 4-50
tuning example 33-58
vp class entry 9-29
NetWare file server 4-40
Network
interface card, configuring 4-19
Network communication
using IPX/SPX 4-40, 4-42, 4-57
using TCP/IP 4-34, 4-41
Network connection
how implemented 9-30
types of 9-28
when to use 4-8
Network Information Service 4-17
Network interface cards
and listen threads 9-34
sqlhosts file 9-34
using multiple 9-34
Network security
files 4-22
hosts.equiv 4-22
.netrc file 4-23
Network virtual processors
description of 9-28
how many 9-29
NIS servers, effect on /etc/hosts and /etc/services 4-18
NOAGE parameter
description of 33-60
purpose of 9-22
Node, index (disk)
branch node defined 38-50
checking horizontal and vertical nodes 35-11
contents of leaf nodes 38-52
creation of branch nodes 38-52, 38-53
defined 38-50
leaf node defined 38-50
pointer 38-52
root node 38-51
root node defined 38-50
types of 38-50
what branch nodes point to 38-53
when root node fills 38-51
Non-Informix databases 1-11
NTFS files 13-6
null file
for creating a chunk 14-8
NUMAIOVPS parameter
description of 33-61
purpose of 9-27
Number of
page-cleaner threads 33-14
NUMCPUVPS parameter
and poll threads 9-29
description of 33-62
purpose of 9-20
used by DS_TOTAL_MEMORY 33-30

O

Offset
definition of 13-9
purpose of 14-5
use in prevention of overwriting partition information 13-10

use in subdividing partitions 13-10
when needed 14-5
OFF_RECVRY_THREADS parameter, description of 3-25, 33-63
OLTP application
characteristics of 15-13
definition of 15-13
uses of 15-13
ON-Archive
backing up logical-log files 18-17
configuration file 3-29
config.arc file 3-29
estimating disk space for 13-32
modifying database logging status 17-5
use in starting data replication 26-11
oncatlgr utility
in UNIX startup script 3-32
message file A-7
multiple residency 6-8
oncatlgr.out.pidnum file A-7
oncfg_servername.servernum file 8-10, A-8
oncheck utility
and locking 35-7
blobpage information 14-22
check-and-repair options 35-5
comparison with onstat 29-12
corrective actions 27-4
description of 35-5
list of functions 35-7
obtaining blobspace information 29-68
obtaining blobspace storage statistics 29-67
obtaining chunk information 29-56, 29-57
obtaining configuration information 29-15
obtaining extent information 29-60
obtaining fragmentation information 29-60
obtaining logical-log information 29-46
obtaining physical-log information 29-49
obtaining tblspace information 29-62
options
-cc 35-9
-cd 35-10
-ce 35-11
-cI 35-12
-ci 35-11
-cr 35-12
-pB 14-21, 35-13
-pD 35-14
-pd 35-13
-pe 38-5
-pK 35-14
-pk 35-14
-pL 35-14
-pl 35-14
-pP 35-15
-pp 35-15
-pr 35-15
-pT 35-16
-pt 35-16
overview of functionality 35-5
syntax 35-8
use in consistency checking 27-4
ONCONFIG configuration file
conventions 33-5
conventions used 33-6
description A-7
during initialization 8-6
editing with multiple residency 6-5
mentioned 4-50
multiple residency 4-59, 6-5
parameters 4-51
white space 33-6
ONCONFIG environment variable 3-9
and ONCONFIG file A-7
changes for multiple residency 6-5
interaction with TBCONFIG A-7
multiple residency startup script 6-9
multiple versions of database server 3-32
use with multiple residency 5-5
onconfig file A-8
ONCONFIG file parameters. *See* Configuration parameter.
ONCONFIG parameter, and multiple residency 6-4
onconfig.std file
and multiple residency 6-5
description A-8
during initialization 8-6
sample A-10
ondblog utility 35-18
oninit utility
bringing database server on-line 14-11
option descriptions 35-22
-p option 8-10, 13-25
starting database server 35-20
temporary tables 13-25, 35-21
On-line manuals Intro-13
On-line transaction processing. *See* OLTP application.
On-line transaction processing. *See* PDQ.
onload utility. *See Informix Migration Guide.*
onlog utility
description of 35-23
filters for displaying logical-log records 35-26
filters for reading logical-log records 35-25
reconstructing a global transaction 31-7
onmode utility
adding a shared-memory segment 35-37
changing database server mode 35-30
changing shared-memory residency 12-14, 35-32
decision-support parameters, changing 35-39
description of 35-28
dropping CPU virtual processors 10-10
forcing a checkpoint 12-16, 35-32
freeing a logical-log file 19-14, 19-15
killing a participant thread 30-19

killing a session 30-22, 35-33
killing a transaction 30-19, 30-27, 31-4
options
-c force checkpoint 35-32
-d set data replication type 35-35
-I B-1
-k take off-line 35-30, 35-31
-l switch logical-log file 35-33
-n end forced residency 35-32
-p add or remove virtual processor 35-38
-r begin forced residency 35-32
-R regenerate .infos file 35-38
-s take to quiescent 35-30, 35-31
-Z kill transaction 35-34
set decision-support parameters 35-39
setting database server type 26-11, 26-21, 35-35
switching logical-log files 19-15, 35-33
trapping errors B-1
user thread services onmode utility requests 9-7
ON-Monitor
adding a logical-log file 19-5
adding mirror chunks 24-9
archive menu and options 32-10
data-replication screen 32-15
dbspaces menu and options 32-9
diagnostics screen 32-16
dropping a logical-log file 19-7
enabling mirroring 24-5
force-ckpt option 32-9, 32-10
help 32-4
initialization screen 32-12
logical-logs menu and options 32-11
mode menu and options 32-10
parameters menu and options 32-7
parameters screens 32-12 to 32-17
PDQ screen 32-17
performance screen 32-14
recovering a chunk 24-11
setting performance options 12-12
setting shared memory parameters 12-7, 12-10
setting virtual processor parameters 10-5
starting mirroring 24-9
status menu and options 32-7
taking a chunk down 24-10
onparams utility
adding a logical-log file 19-5, 35-44
changing physical log size, location 35-46
changing physical-log location 21-6
changing physical-log size 21-6
description of 35-43
dropping a logical-log file 19-7, 35-45
onperf utility 29-13
onspaces utility
adding a chunk 35-53
adding a mirror chunk 24-9
changing chunk status 35-59
creating a blobspace, dbspace, or temporary dbspace 35-49
dropping a blobspace or dbspace 35-52
dropping a chunk 35-55
ending mirroring 24-13, 35-59
recovering a down chunk 24-11
starting mirroring 24-9, 35-57
taking a chunk down 24-11
onstat utility
and CPU virtual processors 9-21
comparison with oncheck utility 29-12
description of 35-62
freeing blobpages and timing 35-74
header 35-68
monitoring blobspace 29-64
monitoring buffer use 11-20, 29-20, 29-21, 29-22
monitoring buffer-pool 29-24, 29-25
monitoring byte locks 11-29
monitoring checkpoints 29-16
monitoring chunk status 29-54
monitoring configuration 29-14
monitoring data replication 29-71
monitoring database server profile 29-19
monitoring disk usage 29-55
monitoring fragment load 29-55
monitoring latches 29-27, 29-28
monitoring locks 29-29, 29-30
monitoring log buffers 29-49
monitoring logical-log files 29-45
monitoring physical log 29-48
monitoring sessions 29-35
monitoring shared memory 29-18
monitoring tblspaces 29-34
monitoring transactions 29-41
monitoring virtual processors 29-33, 29-34
option descriptions 35-9, 35-69
options
-a 35-70
-b 35-70
-c 35-65
-D 35-74
-d 35-72, 35-73
-F 35-75
-k 35-79
-l 35-80
-m 35-66, 35-82
-o 35-66
-p 35-84
-R 35-87
-r 35-66
-s 35-88
-t 35-89
-u 35-90
-X 35-94
-z 35-94
-- 35-65, 35-70
(none) 35-69
repeated execution with -r 35-66
repeated execution with seconds parameter 35-68
syntax 35-64
table of options and functions 35-62
terminating interactive mode 35-79
terminating repeating sequence 35-79

tracking a global transaction 30-28
using SMI tables for onstat information 34-33
using with shared-memory source file 35-67
ontape utility
backing up logical-log files 18-17
data replication functions 35-97
description of 35-95
exit codes 35-97
LTAPEBLK, use of 33-46, 33-47, 33-48, 33-86, 33-87, 33-89
modifying database logging status 17-7
tasks performed by 35-95
use in starting data replication 26-11
onunload utility
LTAPEBLK, use of 33-46, 33-47, 33-48, 33-86, 33-87, 33-89
onunload utility. *See Informix Migration Guide.*
ON_RECVRY_THREADS parameter
description of 3-25, 33-64
role in data replication 25-14
OPCACHEMAX parameter
description of 3-28, 33-66
Operating database server, things to avoid 28-3
Operating system, scheduled jobs 3-34
Operating-system files. *See* Cooked file space.
Operating-system kernel parameters, initial values 3-29
Operating-system parameters
description of 12-3
oper_deflt.arc file
and multiple residency 6-8
contents A-9
OPTCOMPIND environment variable
See also OPTCOMPIND parameter.
OPTCOMPIND parameter
description of 33-67
See also OPTCOMPIND environment variable.
Optical storage
and STAGEBLOB parameter 3-28, 33-85
Optical Subsystem memory cache
allocation 29-65, 35-82
availability of memory 29-65, 35-82
described 29-65, 35-66, 35-82
kilobytes of TEXT and BYTE data written 29-65, 29-66
kilobytes of TEXT and BYTE data written to staging area 35-83
number of kilobytes 29-65, 35-83
number of kilobytes of TEXT and BYTE objects written 29-66
number of objects written 29-65, 29-66, 35-82, 35-83
number of TEXT and BYTE objects written to staging area 35-83
session ID for user 29-66, 35-83
size 29-65, 35-82
size of TEXT and BYTE data 35-83
user ID of client 29-66, 35-83
Optical (OPT) virtual processor 9-35
Optimizing, hash joins versus nested-loop joins 33-68
options field
buffer-size option 4-44
keep-alive option 4-48
list of options 4-43
security option 4-49
syntax rules 4-44
ORDBMS. *See* Object-relational database management system.
Outer join, effect on PDQ 15-23

P

Page
bit-map page 38-64
blobspace blobpage 38-64
blobspace free-map page 38-64
components of dbspace page 38-34
compression 38-38, 38-46
dbspace blobpage 38-62
dbspace page types 38-29
definition of full page 38-41
description of 13-10
determining database server page size 14-17
displaying contents with oncheck 35-15
free page, definition of 38-29
fullness bit values 38-26
fullness, 4-bit values 38-27
header components 38-35
least-recently used 11-38
locating in shared memory 11-44
logical page number 38-38
most-recently used 11-38
page types in extent 38-29
physical page number 38-38
relationship to chunk 13-11
size recorded in reserved pages 38-8
slot table 38-35
structure and storage of 38-34
validating consistency 27-6
Page compression 38-36, 38-38, 38-46
Page size
shown in ON-Monitor 12-8
shown with onstat -b 35-71
stored in reserved pages 38-8
Page-cleaner table
description of 11-29
number of entries 11-29
Page-cleaner threads
alerted during foreground write 11-52
codes for activity state 35-75
description of 11-46

flushing buffer pool 11-46
flushing of regular buffers 11-47
monitoring 11-30
monitoring activity 35-65, 35-75
number of 33-14
role in chunk write 11-53
sleeping forever 11-51
PAGE_1CKPT reserved page 38-7
PAGE_1PCHUNK reserved page 38-11
PAGE_2CKPT reserved page 38-7
PAGE_ARCH reserved page 38-8, 38-14
PAGE_CKPT reserved page 38-9
PAGE_CONFIG reserved page 35-12, 38-8
mentioned 8-7, 8-9
PAGE_DBSP reserved page 38-11
PAGE_MCHUNK reserved page 38-13
PAGE_PCHUNK reserved page 38-11
PAGE_PZERO reserved page
contents of 38-8
mentioned 8-8
when written to 38-7
Parallel Database Query. *See* PDQ.
Parallel processing
mentioned 1-7
virtual processors 9-10
Parallel sort
memory allocation 11-34
Parallelism
degree of, definition 15-11
Parameters
setting decision-support with onmode 35-39
specifying dataskip with onspaces 35-61
used with PDQ 15-16
Participant database server
automatic recovery 30-14
description of 30-5
Partnum field in systables 38-21
PATH environment variable 3-8
in shutdown script 3-33
in startup script 3-32
multiple residency startup script 6-9
PDQ (Parallel Database Query)
and remote tables 15-23
and stored procedures 15-22
and triggers 15-22
degree of parallelism 15-11
description of 15-10
DS_MAX_QUERIES parameter 33-26
DS_MAX_SCANS parameter 33-27
DS_TOTAL_MEMORY parameter 3-23, 33-29
effect of table fragmentation 15-3
how used 15-18
MAX_PDQPRIORITY parameter 33-51
monitoring resources 29-40
parameters used to control 15-16
PDQPRIORITY parameter 15-17, 33-51
principal components 15-11
priority
effect of remote database 15-23
queries that do not use PDQ 15-21
queries that use PDQ 15-18
resource allocation 15-16
SET PDQPRIORITY statement 15-17
statements affected by PDQ 15-21
used with fragmentation 15-5
uses of 15-3
when to use 15-12
PDQPRIORITY environment variable
values of 15-17
PDQPRIORITY parameter
and MAX_PDQPRIORITY 33-51
effect of outer joins 15-23
Pending transaction 35-92
Performance
and resident shared-memory 11-17
and shared memory 11-5
and yielding functions 9-14
effect of read-ahead 11-42
effects of VP controlled context switching 9-10
how frequently buffers are flushed 11-37
of CPU virtual processors 9-21
shared-memory connection 4-10
Performance configuration parameters
setting, using a text editor 12-11
setting, using ON-Monitor 12-12
Performance tuning
and extent size 13-40
and foreground writes 11-52
and logical volume managers 13-46
blobspace blobpage size 14-21
disk layout guidelines 13-33
logical-log size 18-5
LRU write 11-52
mechanisms 1-6
minimizing disk head movement 13-39
minimizing disk-head movement 13-39
moving the physical log 21-4
sample disk layout for optimal performance 13-42
spreading data across multiple disks 13-46
tuning amount of data logged 20-5
Permissions, file 14-7
PHYSBUFF parameter
and physical-log buffers 11-24
description of 33-68
PHYSDBS parameter
changing size and location 21-5
description of 33-69
mentioned 3-14, 3-15, 3-16, 3-17, 3-18
where located 20-8
PHYSFILE parameter
changing size and location 21-5
description of 33-70
initial configuration value 20-6
Physical consistency, description of 22-5
Physical log
and virtual processors 9-24
becoming full 20-7
before-image contents 20-5

buffer 20-9
changing size and location
possible methods 21-3
rationale 21-4
restrictions 21-4
using an editor 21-5
using ON-Monitor 21-5
using onparams 21-6
checking consistency 35-12
contiguous space 21-4
description of 20-3
effects of checkpoints on sizing 20-6
effects of frequent updating 20-6
ensuring does not become full 20-7
flushing of buffer 20-10
how emptied 20-11
in root dbspace 38-5
I/O, virtual processors 9-26
managing 21-3
monitoring 29-47
optimal storage of 13-35, 13-36
reinitialize shared memory 21-3
role in fast recovery 22-4, 22-5, 22-6
scenario for filling 20-8
size of 33-70
sizing guidelines 20-5
where located 20-8
Physical logging
and archiving 20-4
and blobspace blobpages 20-5
and data buffer 20-10
and fast recovery 20-4
description of 20-3
details of logging process 20-9
purpose of 20-4
which activity logged 20-4
Physical page number 38-38
Physical units of storage
list of 13-4
Physical-log buffer
amount written 11-50
and checkpoints 20-11
dbspace location 33-69
description of 11-24
events that prompt flushing 11-48
flushing of 11-47, 20-10
mentioned 11-49
monitoring 29-49
number of 11-24
PHYSBUFF parameter 11-24
role in dbspace logging 18-26, 20-10
size of 33-68
when it becomes full 11-49
PIO virtual processors
description of 9-26
how many 9-26
Planning for database server resources 3-3
Platform icons Intro-9
Point-in-time recovery. *See* Backup and Restore Guide.
Point-in-time recovery. *See also* Archive and Backup Guide.
Poll threads
and message queues 11-34
DBSERVERNAME parameter 9-29
description of 9-30
how many 9-29
on CPU or network virtual processors 9-29
Post-decision phase 30-6, 30-9
Precommit phase 30-6
Presumed-abort optimization 30-10, 30-17
Primary database server 25-5
Primary key, use in fragmented table 38-40
Printed manuals Intro-14
Priorities for disk I/O 9-25
Priority aging
description of 9-22
of CPU virtual processors 33-60
preventing 9-22
Private environment file A-6
Privileges
on databases and tables 1-12
Processes
compared to threads 9-5
DSA versus dual process architecture 9-9
that attach to shared-memory 11-11
Processor affinity
AFF_NPROCS parameter 33-9
and AFF_SPROC parameter 33-10
description of 9-12
using 9-22
Processor, locking for multiple or single 33-55
Product icons Intro-9
Profile
displaying counts with onstat utility 35-66, 35-84
monitoring with SMI 34-20
setting counts to zero 35-67, 35-94
Profile statistics 29-19
Program counter and thread data 11-32
Program group
documentation notes Intro-15
release notes Intro-15
PSORT_DBTEMP environment variable
creating temporary implicit tables 13-27
relationship to DBSPACETEMP 33-20
PSORT_NPROCS environment variable, allocating sort memory 11-34

Q

Queues
description of 9-16
disk I/O 9-28
ready 9-17
sleep 9-17
wait 9-18
Quiescent mode
with oninit utility 35-21

R

RAID. *See* Redundant array of inexpensive disks.
Raw device 13-7
  and character-special interface 13-6
  definition of 13-6
Raw disk space 13-6
  compared with cooked space 13-7
  description of 13-6
  how to allocate, UNIX 14-8
  how to allocate, Windows NT 14-9
  on Windows NT 13-7
  rationale for using 13-8
  steps for allocating 13-8
RA_PAGES parameter
  description of 3-21, 33-71
  purpose of 11-42
  reading a page from disk 11-44
RA_THRESHOLD parameter
  description of 3-21, 33-72
  purpose of 11-42
RDBMS. *See* Relational database management system.
Read-ahead
  description of 11-42
  number of pages 3-21, 33-71
  RA_PAGES parameter 11-42
  RA_THRESHOLD parameter 11-42
  threshold for 3-21, 33-72
  when it occurs 11-44
  when used 11-42
Read-only mode, description of 7-4
Ready queue
  description of 9-17
  moving a thread to 9-17, 9-18
Reception buffer 25-9
Recovery
  by two-phase commit protocol 30-10
  fast, description of 22-3
  from media failure 23-5
  parallel processing of 9-10
Recovery mode, description of 7-4
Recovery threads
  off-line 33-63
  on-line 3-25, 33-64
Redundant array of inexpensive disks (RAID), mirroring alternative 23-7
Referential constraints 16-5
regedt32 program
  and sqlhosts registry 4-29
Regular buffers
  and big buffers 11-31
  description of 11-19
  events that prompt flushing 11-47
  how big 11-20
  monitoring status of 11-20
Relay module, version 7.2
  description of 4-60
  example 4-60
  example with three servers 4-62
Release notes Intro-15
Release notes, program item Intro-15
Remainder page, description of 38-41
Remote client 4-22
Remote database, effect on PDQPRIORITY 15-23
Remote hosts 4-22
Replication server. *See* Data replication.
Reserved pages
  archive information 38-8
  checking with oncheck 35-12
  checkpoint information 38-7
  data-replication information 38-8
  dbspace information 38-7
  description of 38-6
  estimating disk space for 13-32
  location in root dbspace 38-5
  optimal storage 13-35
  organization in pairs 38-7
  role in checkpoint processing 38-9
  validating with oncheck 27-5
  viewing of contents 38-8
  when updated 38-7
RESIDENT parameter
  description of 33-73
  during initialization 8-10
  interaction with onmode utility 35-32
  mentioned 12-15
Resident shared memory 8-7, 11-18
  conditions for 11-17
  contents of 11-18
  description of 11-17, 11-18
  internal tables 11-26
  RESIDENT parameter 33-73
  setting configuration parameters 12-7
  turning on / off residency 35-32
Resource allocation
  effects of PDQ 15-16
RESTARTABLE_RESTORE parameter
  description of 33-74
rofferr script Intro-14
Roll back
  in fast recovery 22-8
  mentioned 1-9
Roll forward
  in fast recovery 22-7
  mentioned 1-9
Root dbspace
  and temporary tables 13-19
  calculating size of 13-30
  description of 13-19
  estimating disk space for 13-32
  initial chunk 33-76
  location of logical-log files 18-12
  mirroring 24-7, 33-53
  specified by ROOTNAME parameter 33-75
  structure 38-4
  using a link 33-76
ROOTNAME parameter
  description of 33-75
  mentioned 13-19
  multiple residency 6-6
  used by PHYSDBS 33-69
ROOTOFFSET parameter
  and multiple residency 5-5
  description of 33-75
  mentioned 13-19
  multiple residency 6-6
  when is it needed 14-6

ROOTPATH parameter
and multiple residency 5-5
description of 33-76
mentioned 3-13, 13-19
multiple residency 6-6
setting 3-12
specifying as a link 33-76
ROOTSIZE parameter
description of 33-77
Round-robin distribution scheme 15-6
Row
accommodating large rows 38-44
data row storage 38-40
displaying contents with oncheck 35-13
effects of deletion on index 38-57
effects of modifying 38-44
linking of sections 38-42
maximum in a page 38-36
storage location 38-41
Rowid
as component of index item 38-49
description of 38-37
effect of page compression 38-38
elements of 38-36
for fragmented table 38-39
functions as forward pointer 38-39
locking information derived from 35-80
relation to slot table 38-36
stored in index pages 38-39
structure 38-38
where stored 38-39
RSAM task control block 29-35

S

Sample onconfig.std file A-10
Scalability of Dynamic Server 1-5
Scans
of indexes 11-42
of sequential tables 11-42
parallel processing of 9-10
Scheduled system jobs 3-34
Secondary database server 25-5
Security
database server 1-12
how enforced 1-12
isolating applications 5-4
of database server 1-12
risks with shared-memory communications 4-10
Security options, sqlhosts 4-49
Security option, in options field 4-49
Segment identifier (shared-memory) 11-14
Segment. *See* Chunk.
Semaphore, UNIX parameters
Shared memory semaphore guidelines 12-6
SERVERNUM parameter
and multiple database servers 11-13
and multiple residency 5-4, 5-5, 6-5, 6-6
description of 33-78
how used 11-13
setting 3-12
Server-to-server connections
TCP/IP 4-9
servicename
syntax rules 4-34, 4-40
servicename field in sqlhosts file
choosing an appropriate name 4-40
description of 4-40
with IPX/SPX 4-42
with shared memory 4-40
with stream pipes 4-40
servicename file A-9
servicename.exp file A-9
Services file
on Windows NT 4-20
services file 4-19
multiple residency 6-7
Service, in IPX/SPX 4-42
Session
and active tblspace 11-30
and dictionary cache 11-33
and locks 11-29
and shared memory 11-31
and stored procedure cache 11-34
control block 9-13
description of 9-13
information in SMI tables 34-23, 34-24
monitoring 29-35
primary thread 11-32
shared-memory pool 11-25
sqlexec threads 9-7
threads 9-7
Session control block 9-13
description of 11-31
shared memory 11-31
SET INDEX statement 11-21
SET PDQPRIORITY statement 15-17
SET TABLE statement 11-21
setup program
and sqlhosts registry 4-28
Share lock (buffer), description of 11-36
Shared data 11-5
Shared library files A-5
Shared memory
adding segment with onmode 35-37
allocating 11-25
amount for sorting 11-33
and blobpages 11-63
and critical sections 11-56
and SERVERNUM parameter 11-12
and SHMBASE parameter 11-12
attaching additional segments 11-13, 11-15
attaching to 11-11
base address 33-80
buffer allocation 11-20
buffer locks 11-36
buffer pool 11-19, 20-9
buffer table 11-27
buffer-table hash table 11-27
buffer, frequency of flushing 33-44
changing residency with onmode 12-14, 35-32
changing with onmode 35-39
checkpoint 11-57
chunk table 11-28
communication 4-34
communications portion 11-34

configuring for multiple residency 5-6
connection 4-12
copying to a file 29-18
created during initialization 8-7
database server requirements 11-12
data-replication buffer 25-10
dbspace table 11-28
description of 11-5
dictionary cache 11-33
dumps 3-27, 33-34, 33-35
dynamic management of 1-7
effect of operating system parameters 12-3
examining with SMI 34-5
first segment 11-13
for interprocess communication 11-5
global pool 11-34
hash tables 11-26
header 11-15, 11-19
heaps 11-33
how much 11-9
how utilities attach 11-12
how virtual processors attach 11-12
identifier 11-13
initializing 8-3, 35-20
initializing structures 8-8
internal tables 11-26
key value 11-13
largest allocation of resident portion 11-19
latches 11-35
locating a page 11-44
lock table 11-29
logical-log buffer 11-22
lower boundary address problem 11-15
mirror chunk table 11-28
monitoring 29-18, 35-62
mutexes 11-35
operating-system segments 11-9
page-cleaner table 11-29
performance 11-5
performance options 12-12
physical-log buffer 11-24, 33-68
pools 11-25
portions 11-7
purposes of 11-5
reinitializing 12-14
resident portion 12-7
resident portion, flag 33-73
resident portion, mentioned 8-7
saving copy of with onstat 35-66
segment identifier 11-14
segments, dynamically added, size of 33-79
session control block 11-31
session data 11-31
setting configuration parameters 12-3
SHMADD parameter 11-25
SHMTOTAL parameter 11-9
SHMVIRTSIZE parameter 11-25
size displayed by onstat 11-9, 35-68
size of virtual portion 11-25
sorting 11-33
stacks 11-32
STACKSIZE parameter 11-32
stored procedures cache 11-34
synchronizing buffer flushing 11-46
tables 11-26
tblspace table 11-30
the resident portion 11-18
thread control block 11-32
thread data 11-32
thread isolation and buffer locks 11-36
total size 11-15
transaction table 11-30
use of SERVERNUM parameter 33-78
user table 11-31
virtual portion 11-25, 11-26, 12-10, 12-15
virtual segment, initial size 33-82
Shared-memory buffer, maximum number 33-12
Shared-memory connection
example 4-54
how a client attaches 11-11
in servicename field 4-40
message 9-31
message buffers 11-34
virtual processor 9-28
SHMADD parameter
description of 11-25
specifying value 33-79
SHMBASE parameter
attaching first shared-memory segment 11-13
description of 11-14, 33-80
warning 11-14
shmem file
and assertion failures 27-7
and DUMPSHMEM parameter 33-35
shmem.xxx file A-10
shmkey
attaching additional segments 11-15
description of 11-13
SHMTOTAL parameter
description of 11-9
specifying value 33-81
SHMVIRTSIZE parameter
description of 33-82
specifying size of virtual shared memory 11-25
Shutdown
graceful 7-7
immediate 7-7
mode, description of 7-4
taking off-line 7-8
Shutdown script
multiple residency 6-9
steps to perform 3-32
Single processor computer 9-21
SINGLE_CPU_VP parameter
and single processor computer 9-21, 9-22
description of 33-83
Situations to avoid 28-3
Sizing guidelines
logical log 18-6
physical log 20-5
Sleep queues, description of 9-17
Sleeping threads
forever 9-18
types of 9-17

Slot table
description of 38-35
entry number 38-35
entry reflects changes in row size 38-38, 38-44
location on a dbspace page 38-34
relation to rowid 38-36
SMI table
aborted table build 8-11
description of 34-5
during initialization 8-11
list of supported tables 34-8
monitoring buffer use 29-23
monitoring buffer-pool 29-26
monitoring checkpoints 29-17
monitoring chunks 29-59
monitoring data replication 29-73
monitoring databases 29-44
monitoring fragmentation 29-63
monitoring latches 29-28
monitoring locks 29-31
monitoring log buffer use 29-51
monitoring logical-log files 29-47
monitoring sessions 29-39
monitoring shared memory 29-20
monitoring virtual processors 29-35
preparation during initialization 8-11
See also sysmaster database.
See also System monitoring interface.
Sockets
on Windows NT 4-9
Sockets, in nettype field 4-32
Software dependencies Intro-4
Sorting
and shared memory 11-33
as parallel process 9-10
Split read 23-9
SQL statement
using temporary disk space 13-25
SQLCA
warning flag when fragment skipped during query 33-16
SQLEXEC environment variable, example 4-61
sqlexec thread
and client application 9-13
as user thread 9-7
role in client/server connection 9-31
sqlhosts file 3-28
and communications support module 4-15
communications support module option 4-45
description of 4-27
mentioned 1-5
sqlhosts file and registry
and multiplexed connection 4-7
buffer size option 4-45
dbservername field 4-31
defining multiple network addresses 9-33
end of group option 4-46
entries for multiple interface cards 9-34
fields of 4-30
group option 4-46
identifier option 4-48
keep alive option 4-48
local loopback example 4-55
mentioned A-10
multiple connection types, example 4-52
multiple dbservernames 33-17
multiple residency 6-7
nettype field 4-31
network connection example 4-56
options field 4-43
security options 4-49
servicename field 4-40
shared-memory example 4-54
syntax rules 4-30
sqlhosts registry
central registry 4-29
changing 4-29
description of 4-28
INFORMIXSQLHOSTS environment variable 4-29
location of 4-29
SQLRM environment variable 4-61
SQLRMDIR environment variable 4-61
Stack
and thread control block 9-16
description of 9-15
INFORMIXSTACKSIZE environment variable 11-32
monitoring stack size 29-38
pointer 9-16
size of 11-32
STACKSIZE parameter 11-32
thread 11-32
STACKSIZE parameter
changing the stack size 11-32
description of 33-84
STAGEBLOB parameter 33-85
Standard database server 25-5
Starting database server
and initializing disk space 3-30
using oninit 35-20
Startup script
multiple residency 6-9
multiple versions of database server 3-32
Statement, SET PDQPRIORITY 15-17
Statistics. *See* onstat utility.
Stored procedure
effect of PDQ 15-22
Stored procedures cache 11-34
stores7 database Intro-5
Stream-pipe connection
advantages and disadvantages 4-11
in servicename field 4-40
Structured Query Language 1-4
INSERT statement 15-6
Structured Query Language.
See also SQL statement.
Swapping memory 11-17
Switching between threads 9-16
Symbolic link
using with shared libraries A-5
Symbolic link, using with TAPEDEV 33-87
Symmetric multiprocessing, description of 9-3
Sysmaster database
description 34-3
See also System monitoring interface.

sysmaster database
See also SMI table.
buildsmi.xxx file A-4
description 34-3
failure to build A-4
functionality of 34-3
list of topics covered by 34-5
SMI tables 34-5
space required to build 34-4
types of tables 34-3
warning 34-4
when created 34-4
System catalog tables
and dictionary cache 11-33
and oncheck 35-9
disk space allocation for 38-68
how tracked 38-69
location of 13-22
optimal storage of 13-36
sysfragments table 15-4, 38-22
tracking a new database 38-69
tracking a new table 38-70
validating with oncheck 27-5
System failure, defined 22-4
System monitoring interface (SMI)
See also SMI table.
accessing SMI tables 34-6
and locking 34-7
and SPL 34-7
and triggers 34-7
description 34-3
SMI tables map 34-30
tables
list of supported 34-8
sysadtinfo 34-9
sysaudit 34-10
syschkio 34-11
syschunks 34-11
sysconfig 34-13
sysdatabases 34-14
sysdbslocale 34-15
sysdbspaces 34-15
sysdri 34-17
sysextents 34-18
syslocks 34-18
syslogs 34-19
sysprofile 34-20
sysptprof 34-23
syssesprof 34-24
syssessions 34-26
sysseswts 34-28
systabnames 34-29
sysvpprof 34-30
using SELECT statements 34-6
using to monitor database server 29-12
using to obtain onstat information 34-33
viewing tables with dbaccess 34-6
System startup script, multiple residency 6-9
System timer 9-17

T

Table
creating temporary 15-8
creating, what happens on disk 38-68, 38-69
description of 13-23
disk-layout guidelines 13-36
displaying allocation information 35-16
high-use 13-39
identifying its dbspace 38-22
isolating 13-37
migration. *See Informix Migration Guide.*
monitoring with SMI 34-29
placing in a specific dbspace 13-23
pseudotables 34-5
purpose of 13-23
recommendations for storage 13-39
relationship to extent 13-23
remote, used with PDQ 15-23
SMI tables 34-5
storage on middle partition of disk 13-39, 13-44
temporary
cleanup during shared-memory initialization 13-25
effects of creating 38-72
estimating disk space for 13-31
message reporting cleanup 36-16
storage of explicit 13-26
storage of implicit 13-27
Tape device
block size 33-46
Tape management 3-33
TAPEDEV parameter
setting 3-12
using a symbolic link 33-87
TBCONFIG environment variable A-7
Tblspace
description of 13-27
displaying information with onstat 35-67, 35-89
for table fragment 38-22
identifying its dbspace 38-22
monitoring active tblspaces 29-32
monitoring with SMI 34-23
number 38-21
number displayed 35-89
number elements 38-22
temporary tblspace during initialization 8-10
types of pages contained in 13-27
Tblspace number
components of 38-22
description of 38-21
displaying with onstat -t 35-89
for table fragment 38-22
includes dbspace number 38-21
retrieving it from systables 38-21
Tblspace table
contents of 11-30
description of 11-30
Tblspace tblspace
bit-map page 38-23
description of 38-19
entries 38-20
location in a chunk 38-15
location in root dbspace 38-5
size 38-23
structure and function 38-19
tracking new tables 38-70
TCP/IP communication protocol
and Windows sockets 4-9
for client-server communication 4-9
in hostname field 4-34
in servicename field 4-41

listen port number 4-42
network software 4-19
using 4-21, 4-22, 4-23
using a wildcard 4-36
using the internet IP address 4-35
using the TCP listen port number 4-42
TEMP TABLE clause, of CREATE TABLE 15-8
Template for ONCONFIG file A-8
Temporary dbspace
and data replication 25-38, 26-7
and DBSPACETEMP 13-26
described 13-20
Temporary disk space
amount required for temporary tables 13-31
operations requiring 13-25
Temporary table
and fragmentation 15-8
DBSPACETEMP parameter 33-19
during initialization 8-10
explicit 13-24
implicit 13-24
rules for use 33-19
where stored 13-26
with oninit utility 35-21
TERMINFO environment variable 3-10
Testing environment 5-4
TEXT and BYTE data
absence of scanning or compression 1-13
buffers 11-64
creating in a blobspace 11-65
descriptor 11-61, 11-64
descriptor, description of 38-37, 38-61
descriptor, time stamp associated with 11-61
effect of Committed Read isolation 11-62
effect of Dirty Read isolation 11-62
effect of modifying on storage 38-62
entering 1-13
how stored 38-59
how updated 38-62
illustration of blobspace storage 11-65
monitoring in a dbspace 29-69
role of descriptor 38-62
role of descriptor in storage 38-61
role of time stamps 11-61
size limitations 38-62
storage on disk 38-61
when modified 38-62
when written 38-61
writing to a blobspace 11-63
writing to disk 11-64
TEXT data type
requires 4-bit bit map 38-26, 38-29
Text editor
setting performance configuration parameters 12-13
setting shared memory parameters 12-8, 12-11
setting virtual processor parameters 10-4
Thread
access to resources 9-10
accessing shared buffers 11-37
and heaps 11-33
and stacks 11-32
concurrency control 11-35
context of 9-13
control block 9-13, 11-32
description of 9-5
dynamic allocation of 1-7
for client applications 9-5
for primary session 9-13
for recovery 9-7
how virtual processors service 9-12
internal 9-7, 9-20
kernel asynchronous I/O 9-27
main_loop() 11-58
migrating 9-16
mirroring 9-7
monitoring 29-35
multiple concurrent 9-13
ON-Monitor 9-7
onstat information 35-67, 35-94
page cleaner 9-7, 11-46
relationship to a process 9-5
scheduling and synchronizing 9-13
session 9-7, 9-20
sleeping 9-18, 11-45
supporting data replication 25-13
switching between 9-16
user 9-7
waking up 9-17
yielding 9-13
Thread control block
creation of 11-32
role in context switching 9-14
Time stamp
associated with TEXT and BYTE data 11-61
blobpage pair 11-61
description of 11-60
location
blobspace blobpage 38-65, 38-66
dbspace blobpage 38-63
dbspace page 38-34
mentioned 11-46
on a blobpage 38-66
page-header and page-ending pair 11-60, 38-35
role in
data consistency 11-60
flushing physical-log buffer 11-50
Time-out condition 35-75
TLI. *See* Transport-layer interface.
Transaction
factors which prevent closure 18-19
global
definition of 30-5
determining if implemented consistently 31-4
identification number, GTRID 31-8
tracking 30-28
heterogeneous commit 33-37
kill with onmode -Z 35-34
mixed result 30-25
monitoring 29-41
pending 35-92
piece of work, definition of 30-5
two-phase commit, examples 30-8

Transaction logging
buffered 16-9
definition of 16-8
unbuffered 16-9
when to use 16-10
See also Logging.
Transaction table
description of 11-30
tracking with onstat 11-30
Transport-layer interface
in nettype field 4-32
Trapping errors with onmode B-1
Triggers, effect of PDQ 15-22
Trusted clients, database server and Windows NT domains 4-6
Trusted domain 4-6
Trusted domain, Windows NT 4-6
Tuning
large number of users 33-58
use of NETTYPE parameter 33-57
Two-phase commit
contrast with heterogeneous commit 30-38
Two-phase commit protocol
automatic recovery 30-10
administrator's role 30-10
mechanisms for coordinator recovery 30-10
mechanisms for participant recovery 30-14
race condition 30-17
configuration parameters for 30-37
coordinator definition 30-5
coordinator recovery mechanism 30-11
description of 30-3, 30-4
errors messages for 30-29
global transaction definition 30-5
global transaction identification number 31-8
heuristic decisions
heuristic end-transaction 30-26
heuristic rollback 30-22
types of 30-21
independent action
definition of 30-18
resulting in error condition 30-20
resulting in heuristic decisions 30-21
results of 30-19
what initiates 30-19
logical-log records for 30-30
messages 30-6
participant recovery 30-14
participant, definition of 30-5
piece of work, definition of 30-5
post-decision phase 30-6, 30-9
precommit phase 30-6
presumed-abort optimization 30-10, 30-17
requirements for flushing logical log records 30-32
role of current server 30-5
use of DEADLOCK_TIMEOUT 30-37
use of TXTIMEOUT 30-37
TXTIMEOUT parameter
and onmode -Z 30-27
description of 30-37, 33-90
in two-phase commit protocol 30-37
role in automatic recovery 30-11, 30-15, 30-17
Types of applications 15-12
Types of buffer writes 11-51

U

Unbuffered disk access
compared to buffered 13-6
Unbuffered disk space
in data storage 13-3
Unbuffered logging
flushing the logical-log buffer 11-55
Unbuffered transaction logging. *See* Logging.
Unique ID 18-12
Units of storage 13-3
UNIX devices
creating a link to a pathname 14-9
ownership, permissions on character-special 14-8
UNIX files
ownership, permissions on cooked files 14-7
UNIX link command 14-9
UNIX operating system
directory for on-line files Intro-15
reading error messages Intro-14
UNIX shutdown script 3-32
UPDATE STATISTICS statement, effect of PDQ 15-21
USEOSTIME parameter
description of 33-91
User accounts and Windows NT domains 4-5
User impersonation 4-24
User session
monitoring 29-35
monitoring with SMI 34-26
status codes 35-90
User table
description of 11-31
maximum number of entries 11-31
User thread
acquiring a buffer 11-43
description of 9-7
in critical sections 11-56
monitoring 11-31
tracking 11-31
Utilities
attaching to shared-memory 11-12
gcore 3-27, 33-32, 33-34
oncheck 35-5
ondblog 35-18
oninit 35-20
onlog 35-23
onmode 35-28, 35-39
onparams 35-43
onstat 35-62
ontape 35-95
-V option 35-4
Utility
finderr Intro-14
Informix Find Error Intro-14
rofferr Intro-14

V

-V option 35-4
VARCHAR data type
  byte locks 11-29
  implications for data row storage 38-40
  indexing considerations 38-59
  requires 4-bit bit map 38-26, 38-29
  storage considerations 38-37
Version, connecting to different 4-60
Virtual portion (shared memory)
  adding a segment 12-15
  contents of 11-25, 11-26
  global pool 11-34
  setting configuration parameters 12-10
  size of 11-25
  stacks 11-32
  stored procedures cache 11-34
Virtual processor
  access to shared memory 11-7
  adding and dropping 9-11
  add/remove with onmode 35-37
  ADM class 9-17
  ADT class 9-36
  advantages 9-9
  AIO class 9-27
  AIO, how many 9-28
  and context switching 9-10
  and ready queue 9-17
  as multithreaded process 9-5
  attaching to shared memory 11-7
  attaching to shared-memory 11-12
  binding to CPUs 9-12
  classes of 9-7, 9-20
  coordination of access to resources 9-10
  CPU class 9-20
  description of 9-5
  disk I/O 9-24
  dropping (CPU) in on-line mode 10-10
  during initialization 8-8
  how threads serviced 9-12
  LIO class 9-24
  LIO, how many 9-25
  logical-log I/O 9-25
  monitoring 29-33
  moving a thread 9-9
  MSC (miscellaneous) class 9-36
  network 9-28, 9-29
  number in AIO class 33-61
  number in CPU class 33-62
  OPT (optical) class 9-35
  parallel processing 9-10
  physical log I/O 9-26
  PIO class 9-24
  PIO, how many 9-26
  priority aging 33-60
  setting configuration parameters 10-3
  sharing processing 9-9
  use of stack 9-15
VP class in NETTYPE parameter 9-29
VP.servername.nnx file A-10

W

Wait queue
  and buffer locks 11-36
  description of 9-18
Waking up threads 9-17
Warning
  buildsmi script 34-5
  files on NIS systems 4-18
  interpreting after running oncheck -cc 27-6
  when fragment skipped during query processing 33-16
White space in ONCONFIG file 33-6
Wildcard addressing
  by a client application 4-38
  example 4-37
  in hostname field 4-36
Windows Internet Name Service, mentioned 4-20
Windows NT
  program groups for on-line notes Intro-15
  reading error messages Intro-14
Windows Sockets 4-9
WORM. *See* Optical devices.
Write types
  chunk write 11-53
  foreground write 11-52
  LRU write 11-52

X

X/Open compliance
  level Intro-16

Y

Yielding threads
  and ready queue 9-17
  at predetermined point 9-14
  description of 9-13
  on some condition 9-14
  switching between 9-12
ypcat hosts command 4-18
ypcat services command 4-18

Symbols

.informix file 3-9
.netrc file 4-23
  sqlhosts security options 4-49
.rhosts file 4-23
  sqlhosts security options 4-49